14

ASIP Design and Prototyping for Wireless Communication Applications

Atif Raza Jafri, Amer Baghdadi and Michel Jezequel
Telecom Bretagne/Lab-STICC CNRS
France

1. Introduction

Last three decades can undoubtedly be said as the decades of wireless communications. New state of the art data processing techniques have been developed and used in practical system. These techniques include advanced Error Control Coding (ECC), Bit Interleaved Coded Modulation (BICM), high ordered Quadrature Amplitude Modulations (QAM), Signal Space Diversity (SSD), Multi Input Multi Output (MIMO), Orthogonal Frequency Division Multiplexing (OFDM), and others.

To guide the wireless communication industry, for the adoption of new developed techniques in the commercial devices, diverse wireless communication standards have been evolved rapidly in the domain of cellular telephone network, local/wide wireless area networks and Digital Video Broadcast (DVB). These standards propose best combination of evolved data processing schemes at the transmitter to protect data from destructive channel effects under different transmission conditions. Taking an example of mobile telephony, by using newly developed concepts such as spatial multiplexing feature of MIMO, the evolving 3GPP-LTE standard is aimed at achieving 100 Mega bits per second to support high data rate real-time applications on a mobile terminal. Similar trends are also visible in other public domains of wireless communication such as DVB-NGH (Next Generation Handheld), which is the next generation standard for video broadcasting, providing services for fixed and mobile terminals. The other aspect under consideration is the global roaming or seamless coverage across geographical regions which demands multi-mode radios compatible with different systems and standards to provide seamless services at fixed location.

While translating these requirements on the physical layer of a radio terminal, this can be seen as a flexible high throughput hardware platform which can be configured to the required air interface. The overall flexibility of the radio platform can be achieved through the flexibility of individual components at transmitter side (encoder, interleaver, mapper, etc.) and at receiver side (equalizer, demapper, deinterleaver, decoder, etc.). This emerging flexibility need in digital baseband design constitutes a new major challenge when added to the ever increasing requirements in terms of throughput and area. In addition to technical requirements associated with rapid growth in wireless communication industry, the severe time-to-market constraints compel the designers for adopting the rapid design, validation and prototyping flow to conceive these wireless radio terminals for their successful and timely delivery in the market. In short, the diverse requirements of wireless communication standards imply, between others, two crucial requirements on hardware implementation: (1)

Hardware platform flexibility for multi-standard support, and (2) Rapid prototyping flow for system validation under different use case scenarios.

1.1 ASIP and rapid design flow

Considering the first requirement of flexibility, the very first idea about the flexible platform was presented in the initial work on Software Defined Radio (SDR) (Mitola, 1995). Any reconfiguration of an SDR platform simply corresponds to a change in a software program. The required software does not even need to be stored in the device itself, since it can be downloaded, thereby bringing easy maintenance capability to the radio. In this proposition, off-the-shelf General Purpose Processors (GPP) and Digital Signal Processors (DSP) were presented as programmable Processing Elements (PE) of different functional block of a flexible radio platform. With increasing demand of high throughput and low power requirements, GPP and DSP are no more suitable due to their limited parallelism and huge flexibility which is more than what is required in PEs of functional blocks of future radio platforms and hence causing low throughput and high power consumption.

In this regard, Application Specific Instruction-set Processors (ASIPs) are increasingly used in complex System on Chip (SoC) designs. ASIPs are tailored to particular applications, thereby combining performance and energy efficiency of dedicated hardware solutions with the flexibility of a programmable solution. The main idea is to design a programmable architecture tailored to a specific application, thus preserving only the required flexibility.

Coming to the second requirement of rapid design flow, while selecting ASIP as the implementation approach, an ASIP design flow integrating hardware generation and corresponding software development tools (assembler, linker, debugger, etc.) is mandatory. In this regard, by looking at available commercial solutions for ASIP design, it is possible to identify three main classes based on the degree of freedom which is left to the designer:

- Architecture Description Language (ADL) based solutions which can be also defined as ASIP-from-scratch. This approach results in the highest flexibility and efficiency, but on the other hand it requires a significant design effort.
- Template architecture based which allow the designer to add custom instructions to a pre-defined and pre-verified core, thus restricting the degree of freedom with respect to the previous approach to the instruction set definition only.
- Software configurable processors and reconfigurable processors with a fixed hardware, including a specific reconfigurable ISE fabric, which allows the designer to build custom instructions after the fabrication.

CoWare Processor Designer is an ASIP design environment entirely based on LISA (Hoffmann et al., 2001). The language syntax provides a high flexibility to describe the instruction set of various processors, such as SIMD (Single-Instruction Multiple-Data), MIMD (Multiple-Instruction Multiple-Data) and VLIW (Very long instruction word)-type architectures. Moreover, processors with complex pipelines can be easily modeled.

Processor Designer's high degree of automation greatly reduces the time for developing the software tool suite and hardware implementation of the processor, which enables designers to focus on architecture exploration and development. The usage of a centralized description of the processor architecture ensures the consistency of the Instruction-Set Simulator (ISS), software development tools (compiler, assembler, and linker etc.) and RTL (Register Transfer Level) implementation, minimizing the verification and debug effort. Using the Processor

Designer's automated design and optimization environment which utilizes LISA language description to develop a wide range of processor architectures, like SIMD and VLIW as well as processors with DSP or RISC-specific features. The generation of the software development environment by Processor designer enables to start application software development prior to silicon availability, thus eliminating a common bottleneck in embedded system development.

1.2 ASIP for wireless communication applications

Consider the system diagram of Fig. 1, where transmitter includes the channel coding, BICM interleaving, constellation mapping and finally a possibility of SSD and MIMO STC. In the channel encoder there are several flexibility parameters such as input bit in encoder, states of the encoder and trellis construction. BICM interleaver, mapper, SSD and MIMO STCs have their own flexibility parameters.

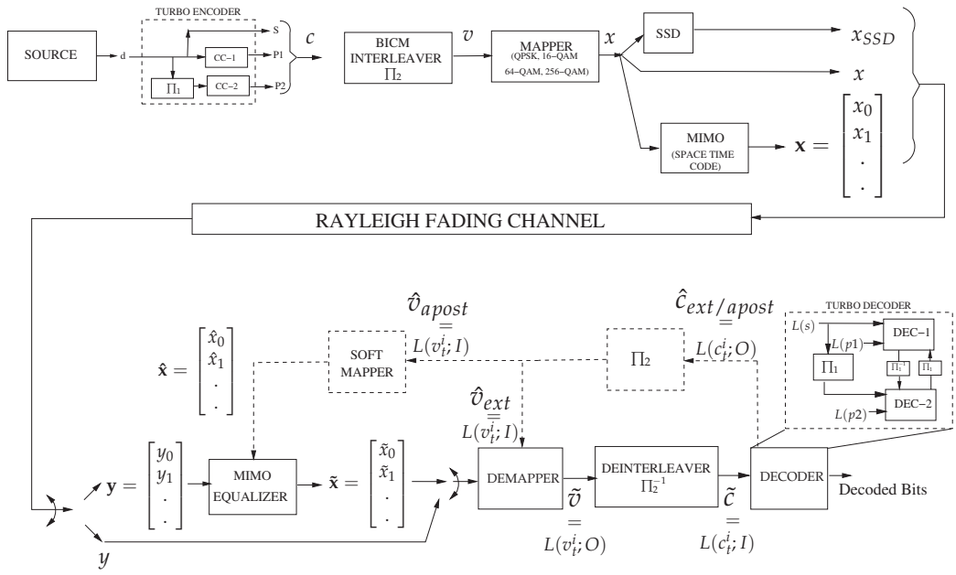


Fig. 1. System Diagram of a Modern Radio Platform

On the receiver side there are blocks such as MIMO equalizer, symbol demapper, deinterleaver and channel decoder. There are also the possibilities of applying iterative/turbo processing to achieve the error rate performance approaching theoretical limits e.g turbo decoding where decoders share the extrinsic information, turbo demodulation where there is an additional feedback exist between decoders and demapper and finally turbo equalization where there is a feedback also to the equalizer. On hardware side, three ASIPs dedicated for turbo decoding, demapping and MIMO equalization functions are required to achieve overall flexibility of the presented receiver. These three ASIPs are named as TurbASIP (Muller et al., 2009), DemASIP (Jafri, Baghdadi & Jezequel, 2009), and EquASIP (Jafri, Karakolah, Baghdadi & Jezequel, 2009). The design methodology adopted to conceive these ASIPs is comprised of four steps:

1. Step 1: Analysis of target application and underlined algorithms with respect to flexibility needs of multi-standard requirements.
2. Step 2: Derivation of architectural choices for the target flexibility and parallelism degree.
3. Step 3: Design of basic building blocks of the ASIP. Efficient resource usage and sharing are considered at this step.

4. Step 4: Design of the complete architecture of the ASIP (including instruction-set, datapath, pipeline stages, memory banks, and I/O interfaces).

To address the severe time to market constraint, a rapid flow for ASIP modeling, validation, and FPGA prototyping is used. This flow is based on Processor Designer Framework from CoWare Inc. The whole process is divided into three abstraction levels where first one is LISA ADL, the second is the HDL level, and finally FPGA/ASIC implementation is last level of this rapid prototyping flow. In this chapter EquASIP is taken as an example to demonstrate the whole approach: (1) Four steps involved in wireless communication ASIP design and (2) Associated three abstraction levels of a rapid validation and FPGA prototyping flow.

2. EquASIP: ASIP-based MMSE-IC linear equalizer

The use of multiple antennas is recognized as a key enabling technology in high performance wireless communications. Most of emerging wireless standards, such as IEEE 802.11m, IEEE 802.16, and 3GPP LTE, propose the use of MIMO systems with different features and parameters. In these standards MIMO techniques such as time diversity and/or spatial multiplexing are specified. Diversity and/or multiplexing achieved through MIMO in different standards are summarized in Table 1. State of the art MIMO detection techniques

MIMO Feature	**IEEE 802.11n**	**IEEE 802.16e**	**3GPP-LTE**
Time Diversity(Alamouti)	✓	✓	✓
Spatial Multiplexing	✓	✓	✓
Golden Code		✓	
Mixed Diversity/ Multiplexing		✓	✓

Table 1. Multi Standard MIMO Support

can be classified in three categories (Burg et al., 2005): Maximum Likelihood (ML) detection, Sphere Decoding (SD), and linear filtering based detection. The complexity of ML detection increases exponentially with the number of antennas and modulation order. The SD approach has a polynomial complexity. To perform SD, first a QR decomposition of channel matrix is carried out and then tree exploration is performed. This tree search is further categorized as depth-first and breadth-first methods. The depth-first has a reduced area complexity and optimal performance, but has variable throughput with SNR. In breath-first case, the most famous algorithm is the K-best in which K best nodes are visited at each level. Hence, the complexity depends on K. A large value of K results in high complexity and good performance. Linear filtering based solutions such as Minimum Mean Squared Error - Interference cancellation (MMSE-IC), considerably reduce the complexity of the hardware implementation of a MIMO detector. Whereas the compensation for sub-optimality can be achieved using an iterative equalization with the channel decoder.

In linear filtering based solution, matrix inversion implying complex numbered operations is the most demanding computational task. Hence, most of the existing work has been focused on the inversion of variable-sized complex-numbered matrices. Matrix inversion based on QR Decomposition Recursive Least Square (QRD-RLS) algorithm has been proposed (Karkooti et al., 2005). In (Myllyla et al., 2005), authors have proposed a Coordinate Rotation Digital Computer (CORDIC) and Squared Givens Rotation (SGR) based Linear MMSE detector while in (Edman & Öwall, 2005) a linear array architecture for SGR implementation has been

introduced. Matrix inversion through block-wise analytical method has been implemented in (Eilert et al., 2007). Two separate MMSE-IC2 equalizers for 4×4 turbo MIMO SM environment using QPSK and QAM-16 modulations, implementing CORDIC method of QR decomposition, have been proposed in (Boher et al., 2008) for fast fading applications. Using analytic method of matrix inversion, a fully dedicated architecture for MMSE-IC1 LE for 2×2 turbo MIMO system with pre-coding used in quasi static channel has been proposed in (Karakolah et al., 2009). The other work carried out in (Kim et al., 2008) shows exciting results in terms of throughput for 802.11n MIMO-OFDM application. The implementation is based on a inverse free architecture using square-root MMSE formulation.
To the best of our knowledge all the available implementations target a specific STC with limited modulation support. In the following sections, the process of developing a flexible MMSE-IC equalizer using the ASIP approach and conforming to multi-standard requirements is explained.

2.1 Step 1: Algorithm analysis and flexibility requirements

At the inputs of the equalizer, as shown in Fig. 1, the received symbol vector $\mathbf{y}$ is given by the following expression.

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \jmath \tag{1}$$

where $\mathbf{y}$ is a vector of size of number of receive antennas (N_r), $\mathbf{x}$ is a vector of size of number of transmit antennas (N_t), $\mathbf{H}$ is channel matrix of size N_rxN_t and the $\jmath$ is column vector of Additive White Gaussian Noise(AWGN) of size N_r. The output $\tilde{x}$ of the MMSE-IC equalizer using time invariant approximation as proposed by (Laot et al., 2005) is given by:

$$\tilde{x}_k = \lambda_k \mathbf{p}_k^H (\mathbf{y} - \mathbf{H}\hat{\mathbf{x}} + \hat{x}_k \mathbf{h}_k) \tag{2}$$

where $k = 1, 2, ...N_t$, $\hat{\mathbf{x}}$ is vector of decoded symbols of size N_t and $\hat{x}_k$ is k^{th} element of this vector, $\mathbf{h}_k$ is k^{th} column of $\mathbf{H}$ matrix and $(.)^H$ is Hermitian operator. The other parameters λ_k, and $\mathbf{p}_k$ are given by:

$$\mathbf{p}_k = \mathbf{E}^{-1}\mathbf{h}_k \tag{3}$$

$$\mathbf{E} = (\sigma_x^2 - \sigma_{\hat{x}}^2)\mathbf{H}\mathbf{H}^H + \sigma_w^2\mathbf{I} \tag{4}$$

where σ_x^2, $\sigma_{\hat{x}}^2$ and σ_w^2 are variances of transmitted symbols, decoded symbols and noise. I is identity matrix.

$$\lambda_k = \frac{\sigma_x^2}{1 + \sigma_{\hat{x}}^2 \beta_k} \; where \; \beta_k = \mathbf{p}_k^H \mathbf{h}_k \tag{5}$$

Equation (2) can be rewritten in the form

$$\begin{aligned} \tilde{x}_k &= \lambda_k \mathbf{p}_k^H (\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}) + \lambda_k \mathbf{p}_k^H \hat{x}_k \mathbf{h}_k \\ &= \lambda_k \mathbf{p}_k^H (\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}) + g_k \hat{x}_k \end{aligned} \tag{6}$$

where $g_k = \lambda_k \beta_k$ is equivalent bias of AWGN noise whose real part is used in demapper.
Once we correlate these equations of MMSE-IC algorithm to the needs of multi-standard requirements, following are the three considered sources in extracting the flexibility parameters:

- MIMO STC supported at the transmitter
- Time diversity of the channel i.e quasi static, block fading and fast fading
- Possibility of iterative equalization in the receiver

2.1.1 MIMO STC

MIMO Spatial multiplexing (SM), Alamouti code and Golden code are the STCs adopted in emerging wireless standards. For MIMO SM with different antenna dimensions, such as 2×2, 3×3 and 4×4, the expressions (3 to 6) can directly be implemented using channel matrix and received vector inputs. Hence, a hardware capable of implementing variable sized complex matrix operation involved in the algorithm can address MIMO SM from 2 to 4 antennas. As far as Golden code and Alamouti code are concerned, MMSE-IC algorithm can be used by applying equivalent channel transformations on the inputs prior to their use. In case of 2×2 Golden code, the equivalent channel transformation is presented in (Cavalec et al., 2008). The idea is to treat two transmitted vectors (each having two elements) as one transmission of four symbols. By applying equivalent channel transformation, the inputs to the MMSE-IC equalizer are $\mathbf{y}$ of four elements and an equivalent channel matrix $\check{\mathbf{H}}$ of size 4×4. The equivalent channel transformation of Alamouti code is presented in (BOUVET, 2005) which transforms a 2×1 channel matrix into a 2×2 equivalent matrix and 2×2 channel matrix into a 4×4 equivalent matrix. Hence, supporting MIMO SM with an additional capability of equivalent channel transformation, addresses this first source of flexibility parameters.

2.1.2 Time diversity

The time diversity of the channel decides how frequent the computations of equalization coefficients (Equation 3 to Equation 5) is required. For quasi static channel these coefficients are computed once per iteration whereas for fast fading channel they are computed for each received vector per iteration. In case of block fading, these coefficients are computed for a set of received vectors for which channel matrix is considered as constant.

2.1.3 Iterative equalization

The last source of flexibility is the iterative/non-iterative nature of the equalizer. In an iterative context the equalizer must incorporate the *a priori* information.

2.2 Step 2: Architectural choices

In the MMSE-IC algorithm, one can note that the expressions computing equalization coefficients and symbol estimation exhibit similar arithmetic operations. Now considering the flexibility need related to time diversity of the channel, allocating separate resources for equalization coefficients computation will result in an inefficient architecture in case of quasi static and block fading channel. For this reason, and targeting flexibility as well as efficiency, our first architectural choice is based on hardware resource sharing between these two tasks. Out of these two distinctive parts of the algorithm, the one related to equalization coefficient computation is more resource demanding. In fact, in this part of the algorithm, the implied computations can only be done in a serial order. For example, to compute matrix E^{-1} of Equation 3, one need to compute:

- Hermitian of $\mathbf{H}$
- Matrix multiplication $\mathbf{H}\mathbf{H}^{\mathbf{H}}$
- Scaler-Matrix multiplication
- Matrix addition
- Matrix inversion

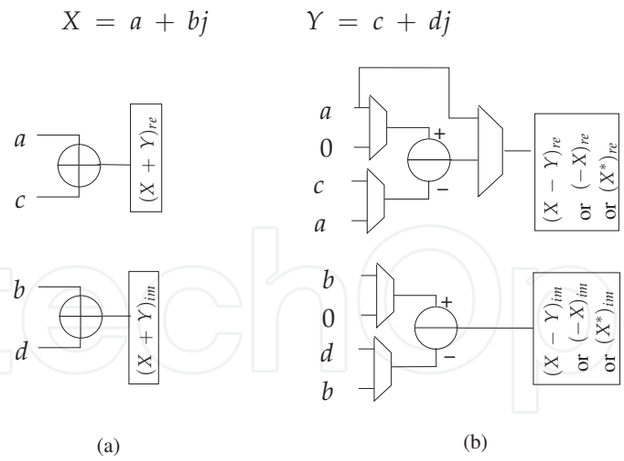


Fig. 2. Basic components (a) Complex adder (b) Complex subtracter, negater and conjugator

The other metrics (such as β_k, λ_k and g_k) are computed with a similar pattern. For this kind of serial computations, temporal parallelism implementation through pipelining can be applied to increase throughput. Now considering the flexibility need related to STC, allocating hardware resources according to the requirements of the most complex STC configuration will result in an inefficient architecture for the low complexity configurations. For this reason, our second architectural choice is based on dimensioning the hardware resources in order to be fully used in all STC configurations. In this regard, the implied complex matrix operations are analyzed and broken down into basic arithmetic operations. Then adequate hardware operators are constructed considering the best tradeoff between flexibility, parallelism and hardware efficiency.

2.3 Step 3: Design of basic building blocks

In this section, a bottom-up presentation approach is adopted to explain the proposed hardware architecture capable of performing complex operations through the basic arithmetic operators. In the first part of this section, we will propose the architectures for the basic complex number operators (performing addition, subtraction, negation, conjugation, inversion) which provide maximum resource sharing. Later on, complex matrix operation, achieved through execution of basic complex number operations (performing multiplication, hermitian and inversion) will be presented.

2.3.1 Complex number operations

In MMSE-IC algorithms, the complex matrix operations can be broken down into basic complex number operation such as addition, subtraction, negation, conjugation and inversion. To perform each operation the architecture of the operator is detailed below.

2.3.1.1 Complex number addition, subtraction, negation and conjugate

The complex number addition needs two real adders whereas a complex numbered subtraction needs two real subtracters. Using two real subtracters, negation of a complex number can be performed. Similarly, conjugate of a complex number, required in calculating the hermitian of a matrix can also share the real subtracter. Fig. 2(a) shows hardware architecture for addition of two complex numbers $X = a + jb$ and $Y = c + jd$ whereas Fig. 2(b)

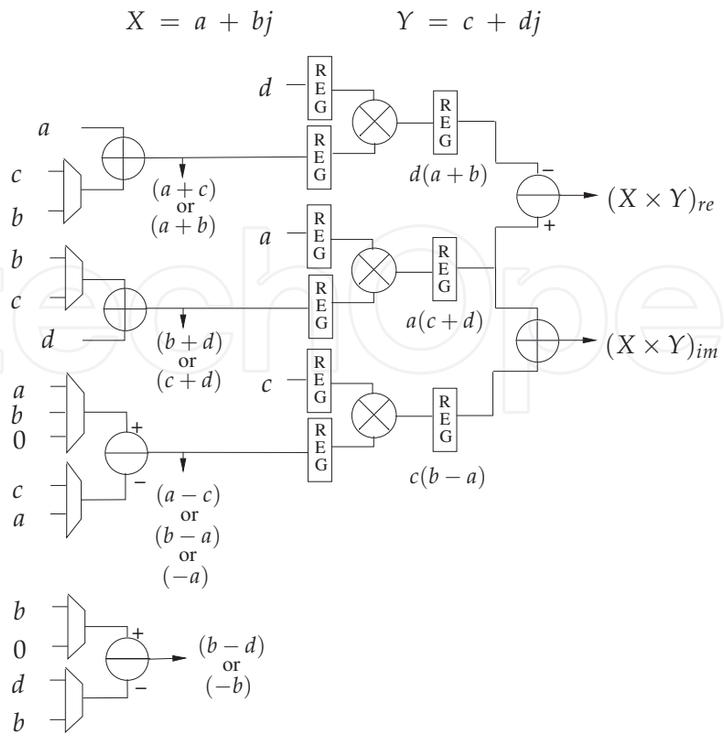


Fig. 3. Combined Complex Adder Subtracter and Multiplier (CCASM)

shows combined architecture of subtraction of X and Y and negation/hermitian of a complex number X.

2.3.1.2 Complex number multiplication

By applying the classical formula (7) of multiplication of complex numbers, a complex numbers multiplier must perform 4 real multiplications and 2 real additions/subtractions.

$$X \times Y = (a + jb)(c + jd) = (ac - bd) + j(ad + bc) \tag{7}$$

A rearrangement may be proposed to reduce the number of multiplications required, as:

$$X \times Y = (a + jb)(c + jd) = a(c + d) - d(a + b) + j\,[a(c + d) + c(b - a)] \tag{8}$$

By applying this reformulation, a complex number multiplier must perform only three real multiplications and 5 real additions/subtractions. Reducing one real multiplier per complex multiplier at the cost of three adders significantly reduces the complexity of the complex number multiplier. In addition the adders and subtracters of first stage of pipelined multipliers can also be used for complex number addition, subtraction, negation and conjugation. A Combined Complex Adder Subtracter and Multiplier (CCASM) is shown in Fig. 3. This architecture is capable of performing all basic operation of complex number addition, subtraction, negation, conjugation (output at first stage of pipeline) and multiplication (output at third stage of pipeline).

2.3.1.3 Complex number inversion

The inverse of a complex number can be computed using following expression:

$$\frac{1}{a+bj} = \frac{a}{a^2+b^2} - \frac{b}{a^2+b^2}j \tag{9}$$

The architecture for this inverter can be obtained by reusing the real multipliers and one adder of the CCASM to compute $a^2 + b^2$. Pre-computed LUT can then be used to find inversion value of $\frac{1}{a^2+b^2}$. Finally, two real multipliers and one subtracter are required for final result computation.

2.3.2 Complex matrix operations

In this subsection we propose the use of basic operators, developed in previous part, to achieve complex numbered matrix operations such as matrix hermitian, multiplication and inversion.

2.3.2.1 Matrix hermitian, addition, subtraction, negation

To perform hermitian operation on a matrix, at first, one need to copy the rows of the matrix into columns of an intermediate matrix. Then by taking complex conjugate of each element of this intermediate matrix, the resultant matrix will be the required hermitian matrix. Using 4 instances of the architecture presented in Fig. 2 with some control logic, provides a fully parallel and flexible architecture to perform Matrix Hermitian, Addition, Subtraction and Negation operations for 2×2 and 4×4 matrices. In case of 3×3 matrix this architecture will be 75% efficient. Hence, to perform any of these operation on 2×2, 3×3 and 4×4 matrices, 1, 3 and 4 clock cycles will be required.

2.3.2.2 Matrix multiplication

To perform a multiplication of two 2×2 matrices, 8 complex multiplications are required whereas for 3×3 and 4×4 matrices the number of complex multiplications required are 27 and 64 respectively. Use of four CCASM (Fig. 3), can efficiently perform all operations (matrix hermitian, addition, subtraction, negation and multiplication) required for 2×2 and 4×4 matrices. For 2×2 matrix multiplications, two complex adders will be required to sum up the multiplication results whereas in 4×4 case, in addition to two complex adders, one more adder will be required. The architecture of 2×2 and 4×4 matrix multiplications is shown in Fig. 4. The number of cycles required to perform 2×2, 3×3 and 4×4 matrix multiplications will be 2, 9 and 16 respectively.

2.3.2.3 Matrix inversion

The matrix inversion can be achieved through one of the following methods:

- based on matrix triangulation
- based on analytical method

The first method based on matrix triangulation can realized using systolic architecture through the LU decomposition, Cholesky decomposition or QR decomposition. The method based on QR decomposition is the most interesting due to its numerical stability and its practical feasibility. It consists of decomposing decompose a matrix $\mathbf{A}$ of size $N \times N$ as $\mathbf{A} = \mathbf{QR}$ where $\mathbf{Q}$ is an orthogonal matrix ($\mathbf{QQ}^H = \mathbf{I}$) and $\mathbf{R}$ an upper triangular matrix. This decomposition allows to compute the inverse of the matrix $\mathbf{A}$ after a simple inversion of the

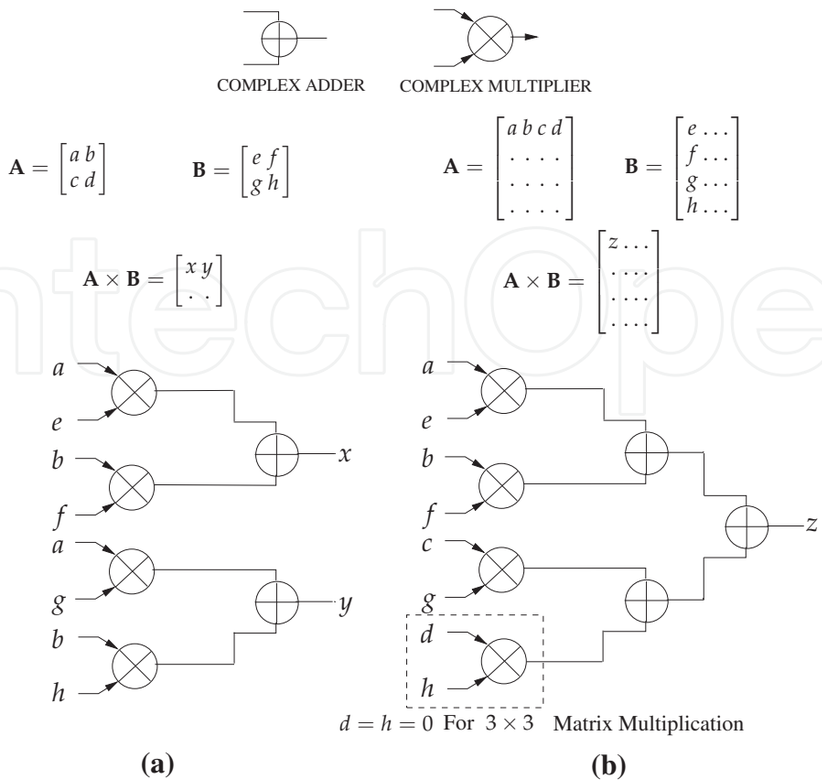


Fig. 4. Complex matrix multiplications (a) 2×2 Matrix multiplication (b) 3×3 and 4×4 Matrix multiplication

triangular matrix $\mathbf{R}$ and a matrix multiplication as $\mathbf{A}^{-1} = \mathbf{R}^{-1}\mathbf{Q}$. There are several methods (Golub & van Van Loan, 1996) to achieve this decomposition, such as the Givens method or the method of Gram-Schmidt. Hardware designers give special attention to the Givens method due to its practical feasibility, its parallelism and its numerical stability (Myllyla et al., 2005)(Edman & Öwall, 2005). The method of Givens consists of triangularization of matrix $\mathbf{A}$ by applying a series of plane rotations called Givens rotations. Each rotation is designed to cancel an element of $\mathbf{A}$. The standard method of Givens uses operations that are not easily implementable, including square root and division. Therefore, there are several variants of this method to avoid these operations. The SGR (Squared Givens Rotations) (Döhler, 1991) and CORDIC method (Volder, 1959) are the best known methods. A comparison between the two approaches: SGR and CORDIC has been made by (Myllyla et al., 2005) through MMSE detector. The results show that the CORDIC-based architecture is more expensive in hardware cost and is 1.5 times slower than those based on SGR. In his thesis work , Edman (Edman, 2006) used SGR method to achieve matrix inversion and studied both triangular and linear architectures. For this type of architecture there are dedicated Processing Elements (PEs) which are used as boundary elements and internal elements of a systolic array or linear array (Edman & Öwall, 2005). Although linear array architecture is flexible for variable sized matrix inversion, it is dedicated to matrix inversion only.

The analytic method of matrix inversion is good candidate, not only for variable sized matrix inversion but also for resource reuse for other matrix computations. The expression for the

inversion of 2×2 matrix through analytical method is given by:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \tag{10}$$

To implement Equation 10 the resources required are a complex number negater and a complex divider. For a 4×4 matrix, the matrix is divided into four 2×2 matrix and inversion can be achieved block wise.

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} W & X \\ Y & Z \end{bmatrix} \tag{11}$$

where

$$\begin{aligned} W &= A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} \\ X &= -A^{-1}B(D - CA^{-1}B)^{-1} \\ Y &= -(D - CA^{-1}B)^{-1}CA^{-1} \\ Z &= (D - CA^{-1}B)^{-1} \end{aligned}$$

The inversion of a 3×3 matrix is performed by extending it to a 4×4 matrix. This can be done by copying all three rows of 3×3 matrix into first three rows of 4×4 matrix and then putting zeros in all elements of fourth row and fourth column where a 1 should be put on the intersection of fourth row and fourth column. The inversion can then be performed using the method mentioned above. The final result lies in first three elements of first three rows (or column). All the expressions involved in the inversion of up to 4×4 matrix can be achieved through already described matrix operations and will be used in the EquASIP.

2.3.2.4 Operator reuse in fixed-point representation

To find the required data width for fixed-point representation of the parameters involved in MMSE-IC algorithm, long simulations have been conducted for all supported system configurations (STC and modulation type). Results analysis have shown that at maximum 16-bit signed representation with different bits for integer and fractional part is sufficient for all the parameters involved during the different computational steps of MMSE-IC LE algorithm. To enable the reuse of hardware resources for these different computations, involving operands with different fixed-point representations, certain rules have been set. First of all, while reading input data from memories, the data which is represented in less than 16-bits, is sign extended to 16-bit. Secondly, a programmable 33 to 16-bit conversion is performed at the outputs of the multipliers. Last of all, to avoid the hazards caused by overflow/underflow during an arithmetic operation, a control mechanism is provided to fix the output at its maximum/minimum limit.

2.4 Step 4: Design of the complete architecture of EquASIP

In this step granularity level of basic building blocks is increased to achieve the data path whereas this datapath is distributed over several pipeline stages to reduce the critical path. Other components such as memory banks are added to achieve complete architecture of the ASIP. The proposed EquASIP architecture is mainly composed of Matrix Register Banks (MRB), Complex Arithmetic Unit (CAU) and Control Unit (CU) besides its memory interfaces. The input to the EquASIP are through "Channel Data Memory" and the soft mapper as

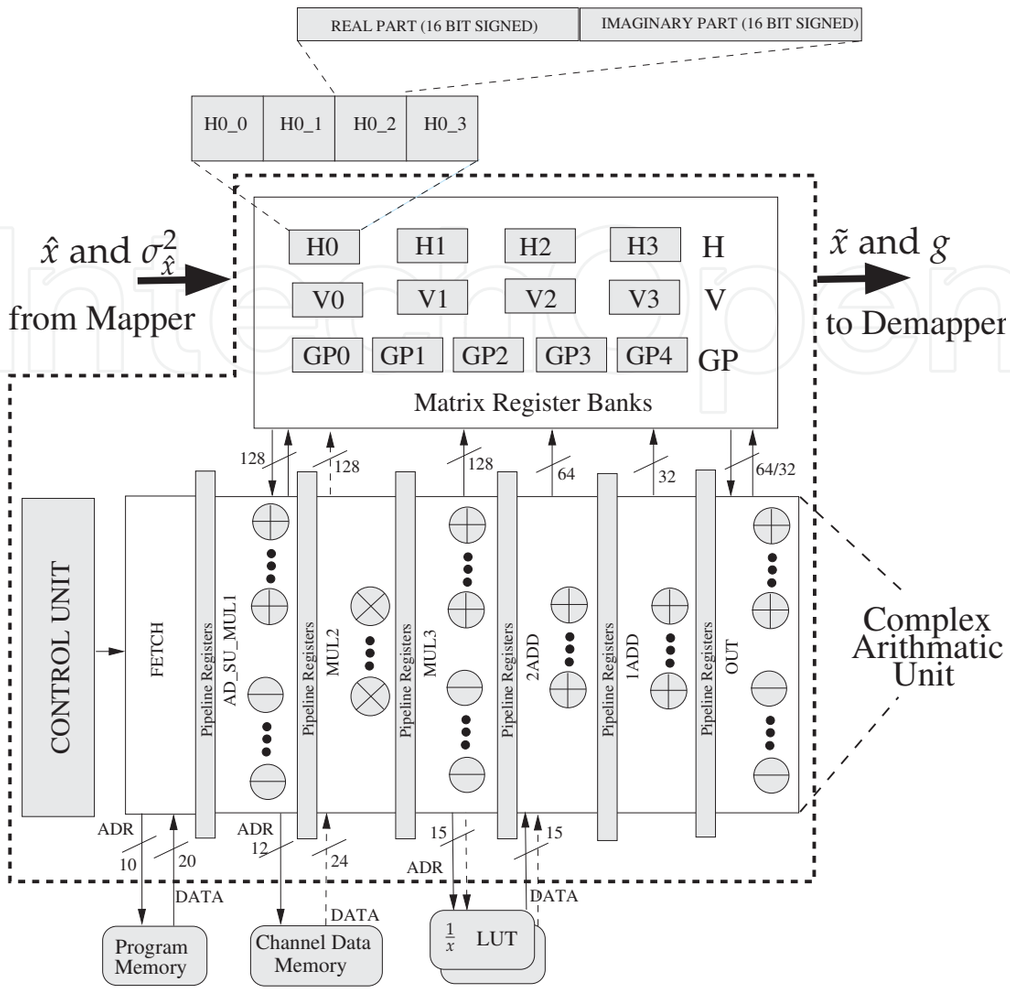


Fig. 5. EquASIP block diagram

shown in Fig. 5. The data bus of all inputs is set to 16 (32 bit for complex number). This provides flexibility to use up to 16 bit data representation and in case of smaller data widths, signed/unsigned extension can be done externally. The ASIP has 7 pipeline stages named as: FETCH, AD_SU_MUL1, MUL2, MUL3, 2ADD, 1ADD and OUT.

2.4.1 Matrix register banks

To store a complex number two separate 16-bit registers have been used, one storing the real and the other imaginary part. Based on the requirements of the Equation 6 for a 4×4 spatially multiplexed MIMO system, 13 MRBs have been proposed, where each MRB can store 4 complex numbers (Fig. 5). H-MRB (H0, H1, H2, and H3) which are connected to the memory, can store 4 rows or columns of Channel Matrix. Four V-MRB (V0, V1, V2, and V3) store 16 entries of $\lambda_k p_k$. GP0, GP1, GP2, GP3 and GP4 are assigned to the storage of g_j , $\hat{x}_j$, y, $g_j\hat{x}_j$ and the estimated symbols $\tilde{x}$ respectively. Other than this specific use, these GP registers save the intermediate results of equalization coefficients. Among other registers there are three registers to store the variances of noise, modulation symbol and decoded symbols besides pipeline registers and the registers for REPEAT instruction.

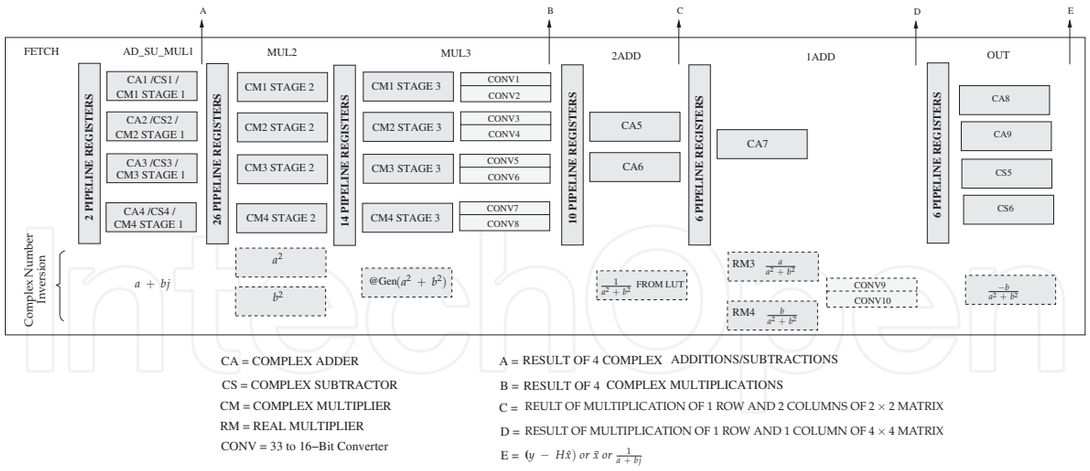


Fig. 6. CAU and pipeline stages

2.4.2 Complex arithmetic unit

The computational resources of the Complex Arithmetic Unit (CAU) of EquASIP are shown in Fig. 6. After fetch pipeline stage, 4 CCASM units (Fig. 3) are spread over three pipeline stages to perform 4 concurrent complex additions, complex subtractions/negation, complex conjugation and complex multiplications. The results of complex addition, subtraction, negation and conjugate operations are copied into destination registers in AD_SU_MUL1 pipeline stage. In MUL3 stage, 33-bit to 16-bit transformation is performed according to the information provided in multiply instruction. The results of four complex multiplication (16-bits for each of real and imaginary part of the complex number) are saved in the target registers. To perform 2×2 matrix multiplication one row/column of first matrix is introduced twice at first input of CCASMs and two columns/rows of second matrix are exposed to the second input of CCASMs. Providing the results of four complex multiplication to two complex adders in 2ADD pipeline stage, the output will give one resultant row/column of multiplication of 2×2 matrix. In case of 4×4 matrix multiplication, one row/column from each matrix goes to the inputs of four CCASM. The results of four multiplications are added together using 2 adders of 2ADD and one adder of 1ADD pipeline stage to output one element of 4×4 matrix multiplication. Complex adders/subtracters in last pipeline stage are used in the computation of Equation 6. The inversion process of a complex number in different pipeline stages is shown as dotted area in Fig. 6. For this particular operation, additional resources are required as Look-Up Tables (LUT), two 33 to 16-bit converters, and two real multipliers.

2.4.3 Control unit

The EquASIP control unit works as administrator of the 7-stage pipelined CAU as mentioned above and shown in (Fig. 5). It controls the flow of the program instructions over the designed datapath (MRBs, CAU) during the different stages of the pipeline. The functioning of the control unit will be reflected during the instruction set presentation which is detailed in the next section.

2.4.4 EquASIP instruction set

The instructions of the proposed ASIP are categorized as follows:

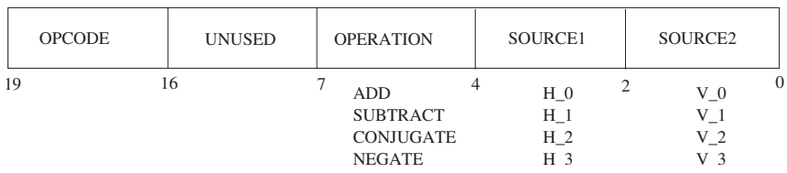


Fig. 7. 20-bit Addition, subtraction, negation and conjugate instructions

2.4.4.1 LOAD, MOVE, REPEAT, NOP

LOAD instruction is used to load channel matrix into H-MRB from memory. While loading data there are possibilities for loading directly or loading after applying conjugation to support equivalent channel transformation. LOAD_CODE instruction is used to initialize the V-MRBs for values which are used in equivalent channel transformation for Golden code. The MOVE instruction is used to transfer data between MRBs whereas REPEAT instruction repeats a block of code as many times as given in REPEAT_SIZE Register. NOP instruction is used to add empty cycles during the execution of the program when required.

2.4.4.2 Matrix addition, subtraction, negation and conjugation instructions

The instruction format for addition, subtraction, negation and conjugation operation is shown in Fig. 7. Besides opcode, the other fields are the "OPERATION" field and two "SOURCE" fields to input two register banks in complex adders and subtracters. The "OPERATION" field of 3-bits indicates the following six different operations: ADD, SUBTRACT, CONJUGATE, NEGATE, MOV_REC and MOV_MOD.

ADD: Using ADD instruction, programmer can select any of the H-MRB as source1 and any of V-MRB as source2. The result of an addition is always saved in GP_0 MRB.

SUBTRACT: Using SUBTRACT instruction, any one of selected H-MRB and any of V-MRB are subtracted and result is always saved in GP_0 MRB.

CONJUGATE/NEGATE: In this single source instruction all four elements of one of the selected H-MRB are conjugated/negated and the results are copied in respective V-MRB i.e V-MRB(n) = Conjugate/Negate(H-MRB(n)) where n can be any integer from 0 to 3.

MOV_REV: This instruction copies the elements of H-MRB(0), in reverse order, into V-MRB(0) with second element in negative form. This is used to align the elements of 2×2 matrix (to be inverted) for a multiplication which results in its determinant. For example if H-MRB(0) has a matrix A with elements a, b, c and d (Equation 10) then V-MRB(0) will have elements $d, -c, b$ and a after the execution of this instruction. To obtain determinant of A ($det(A) = ad - bc$), one can multiply H-MRB(0) with V-MRB(0) and add the results of first two complex multiplications.

MOV_MOD: This instruction is to copy and rearrange the matrix A (saved in H-MRB(0)) in V-MRB(0) to a form required in the inversion of a 2×2 matrix (Equation 10) i.e. if H-MRB(0) has a matrix A with elements a, b, c and d then V-MRB(0) will have elements $d, -b, -c$ and a after the execution of this instruction

2.4.4.3 MULTIPLY

This category is the most demanding one in EquASIP instruction set. Different fields of the multiply instruction are detailed in Fig. 8(a). Eight different opcodes fall under this category to use complex multipliers for multiplication of 4×4 and 2×2 matrices (MULT4X4 and MULT2X2), multiplication of 4 complex numbers (MULT_CMPLX), 3 different MAC instructions (MAC1, MAC2 and MAC3)and two instructions to compute the output symbols

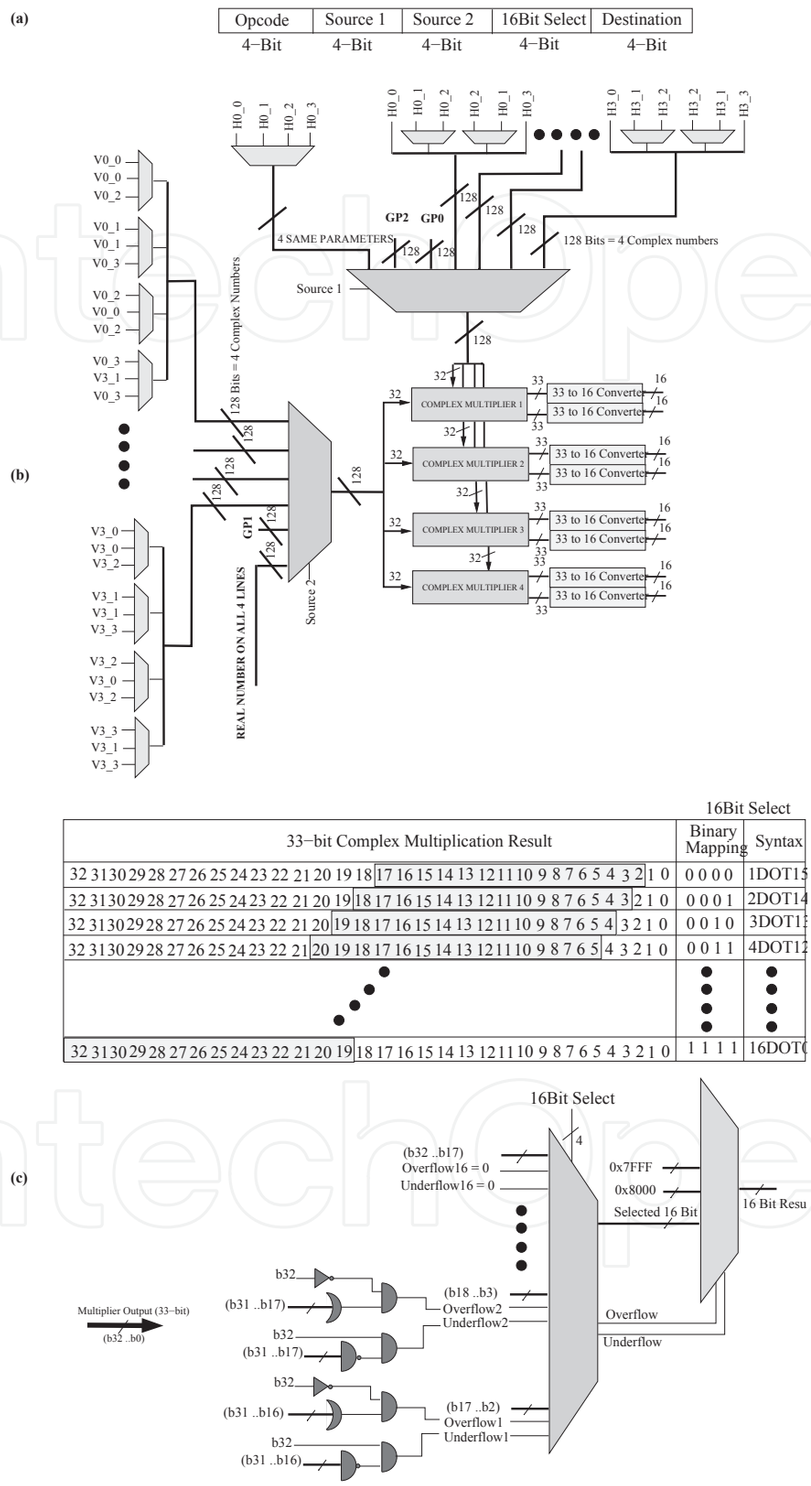


Fig. 8. Complex multiplication datapath: (a) 20-bit Multiply Instruction, (b) Possible inputs to complex multipliers, (c) 33 to 16-bit converter

$\tilde{x}$(OUT1 and OUT2). The 3×3 matrix multiplication is achieved by 4×4 matrix multiplication by providing zero at the input lines of fourth CCASM.
Different possible sources to complex multipliers are shown in the Fig. 8(b). Depending upon the fields "Source1" and "Source2" of the instruction, 4 operands are selected as source1 and 4 as source2 for 4 complex multipliers. To obtain different 16-bit fixed-point representations from 33-bit output of complex multipliers, 33 to 16-bit converters are designed. These converters (Fig. 8(c)) select 16 consecutive bits from 33-bit multiplication result depending upon the "16-Bit Control" field of the instruction. A combinational logic has also been provided to detect overflow/underflow with each choice of bit selection and consequently saturate the output value to maximum/minimum bounds. The "Destination" field of instruction selects the destination for the result.

2.4.4.4 DIVIDE

Two divide instructions have been defined. The first one is the division of a real number while the second one is used to invert a complex number. The first operation during execution of complex number division starts in the third stage of the pipeline to use the real multipliers. LUTs have been used to store the inversion values. The overall operation is shown as dotted area of Fig. 6.

3. Rapid ASIP FPGA prototyping

While selecting ASIP as the implementation approach, an ASIP design flow integrating hardware generation and corresponding software development tools (assembler, linker, debugger, etc.) is mandatory. In this chapter we consider the use of Processor Designer framework from Coware Inc. which enables the designer to describe the ASIP at LISA (Hoffmann et al., 2001) abstraction level and automates the generation of RTL model along with software development tools. ASIP design, validation and prototyping flow has been divided into 3 levels of abstraction as shown in Fig. 9 and is detailed in the following subsections.

3.1 LISA abstraction level

The first step towards the ASIP implementation is the LISA ADL modeling of the proposed architecture and the application program writing (.asm file) to be executed on the ASIP. To simulate the input data memories the contents of these memories, taken from the software reference model of the target application, are written in different sections of the assembly file as defined in the linker command file. With ADL model of the ASIP, Processor Designer framework generates tools like assembler, linker, processor debugger and simulator. Assembler and linker process the application program (.asm file) to generate the executable file (.out file) which is used in Processor Debugger to verify both the ASIP model and the application program. Once the ASIP is verified, a special utility "lvcdgen" can be used to generate Value Change Dump (VCD) file which store all registers content and ASIP output values during the application program execution. The generated VCD file can be used at lower abstraction levels for verification purpose. The "lvcdgen" utility uses Dynamic Simulator Object and executable file of the application to produce this reference VCD file. The complete flow is shown in Fig. 9(a).

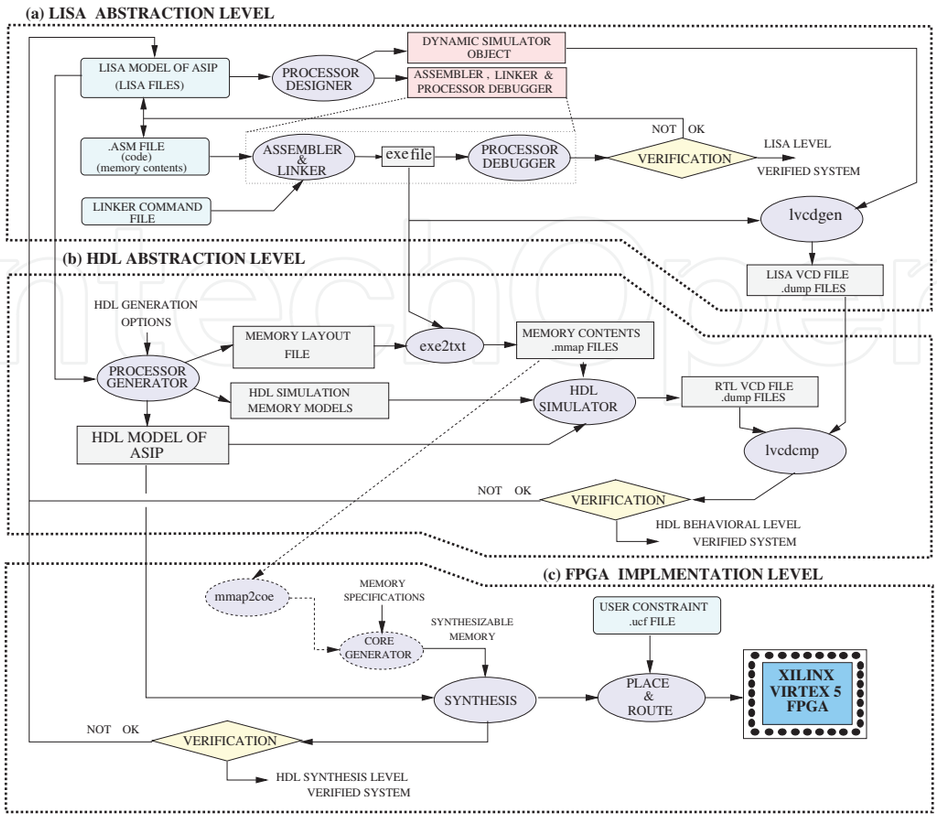


Fig. 9. Prototyping Flow: (a) LISA abstraction level, (b) HDL abstraction level, (c) FPGA implementation level

3.2 HDL abstraction level

Processor Designer framework provides the Processor Generator tool which is configured to generate HDL (VHDL/Verilog) model of the ASIP from LISA model, simulation models of memories and the memory layout file as shown in Fig. 9(b). The quality of the generated HDL depends upon the LISA modeling and the configuration options of Processor Generator. It is highly recommended that LISA modeling should be as close as possible to HDL, e.g if in one pipeline stage we want resource sharing, that resource should be declared once. Otherwise, due to inability to detect sharing, resources will be duplicated in HDL. Other issue is the use of high level operators of LISA which may not be produced by the Processor Generator e.g modulo two operation ("variable % 2" in LISA) should be rather implemented by the LSB manipuation of the considered variable. For memory interface generation, different Memory Interface Definition Files (MIDF) are provided which define the number of ports and latencies. Once memory layout file and executable application program file is available, "exe2bin" utility inputs them to generate the contents of memories in separate .mmap files. With these three inputs (VHDL model, memory model and .mmap files), the VHDL model can be simulated behaviorally using an HDL simulator, e.g ModelSim by Mentor Graphics.

To run HDL simulation, Processor Generator produces ready-to-use Makefile which can be executed to see either the waveforms or to generate VCD file. To verify the generated ASIP HDL model, the VCD file generated through HDL model and the one generated through LISA model (in previous subsection) can be compared using "lvcdcmp" utility.

3.3 FPGA implementation level

At this level, the only missing elements are the synthesizable memory models. Depending upon the FPGA selected, equivalent synthesizable memories are generated through FPGA vendor specific tools and at the same time .mmap memory content files have to be translated, if necessary, in required format for compatibility. With Xilinx devices,"Core Generator" tool can be used to generate the synthesizable memories and "mmaptocoe translator" converts .mmap files into required .coe format. With this complete synthesizable HDL model, synthesis can be performed as shown in Fig. 9(c). After successful synthesis, the placement and routing is performed as per the user constraints file (.ucf file). Inside .ucf file, the user inputs the platform dependent timing and location constraints, e.g the operational frequency and input/output pins. The final step is the generation of the configuration file which can be used to configure the FPGA for the final ASIP prototype model.

3.4 EquASIP FPGA prototyping

On board validation is a crucial step in order to demonstrate the feasibility, resolve any eventual system and/or environment issue, and measure the exact performance of the designed architecture. In our case, a logic emulation board (DN9000K10PCI) integrating 6 Xilinx Virtex 5 devices was available and has been used to validate the designed ASIPs. With this board, appropriate communication controllers are available and can be added to the design in order to read/write various output/input memories from a host computer using a USB interface. Using the Xilinx tool suite ISE, a new project was created integrating the ASIP, corresponding memories, and a board communication controller as shown in Fig. 10. The contents of the input memories i.e Channel Data Memory, $\frac{1}{x}$ LUTs and Mapper Output Memory were generated automatically from the fixed-point software reference model in .coe file format along with a reference result file containing the output of the equalizer. In this prototype, except Channel Data Memory and $\frac{1}{x}$ LUT which are synchronous, rest of the memories are asynchronous. Xilinx Virtex 5 device provides two type of memories, Distributed and Block Memories which can be customized for asynchronous and synchronous respectively. In order to record ASIP's results and to compare them with reference result file, a dual port Equalizer Output Memory has been created. One port of this memory is written with equalization results from EquASIP side and the other port is read by external host computer through USB interface. On this host computer, a graphical user interface with adapted parameters is used in order to setup the various parameters of the board and to download the output memory contents for comparison with reference result file.

4. EquASIP results and performance

By performing hardware synthesis and executing the application programs, performance of EquASIP is ascertained for different configurations and presented below.

4.1 Synthesis results

From the generated RTL description of EquASIP, logic synthesis has been conducted both on ASIC and FPGA. For ASIC target, the processor has been synthesized with Design Compiler tool from Synopsys. For FPGA target, Xilinx ISE tool has been used. In Table 2, the results of synthesis are summarized.

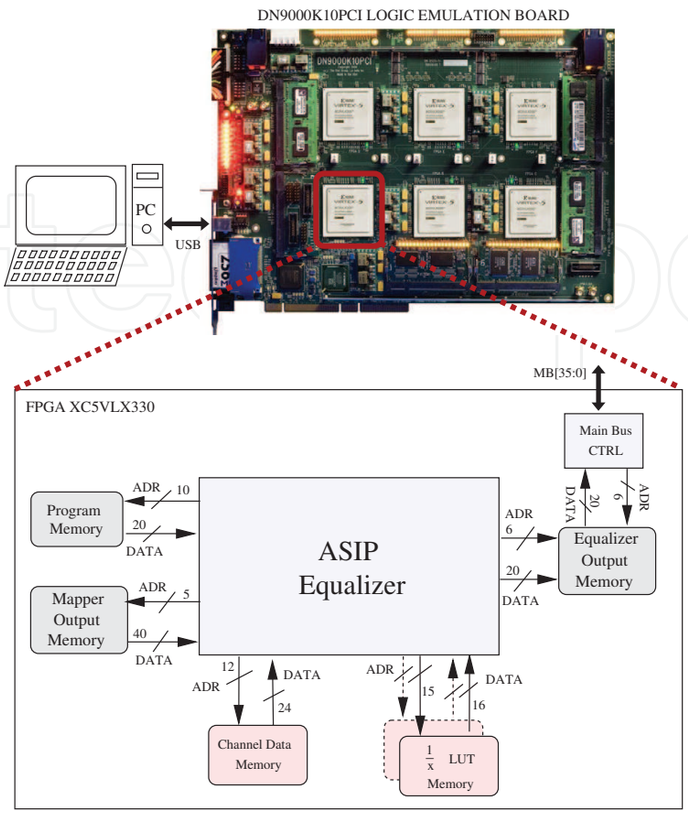


Fig. 10. EquASIP on-board prototype

ASIC Synthesis Results (Synopsis Design Compiler)	
Technology	ST 90nm
Conditions	Worst Case ($0.9V$; 105^oC)
Area	$0.37mm^2$ (84 K Gate)
Frequency	546 MHz
FPGA Synthesis Results(Xilinx Virtex5 xc5vlx330)	
Slice Registers	3,174 out of 207,360 (1%)
Slice LUTs	11,299 out of 207,360 (5%)
DSP48Es	14 out of 192(7%)
Frequency	130 MHz

Table 2. EquASIP synthesis results

4.2 Execution performance

To estimate the throughput of the EquASIP for different system configurations, the number of cycles required to compute the expressions involved in MMSE-IC1 are summarized in Table 3. Using this information, the user can estimate the throughput of the system under different channel's time diversity conditions and used STC. In case of quasi static conditions, after equalization coefficient computation, the throughput in terms of symbols per clock cycle is described in the last row of Table 3. For a 3×3 MIMO SM configuration the symbol throughput is less than a 4×4 MIMO SM. This is due to the fact that for a 3×3 MIMO SM system 25%

Expression	**MIMO 2×2 (Cycles)**	**MIMO 3×3 (Cycles)**	**MIMO 4×4 (Cycles)**
E (Ref. eq. 4)	18	33	50
E^{-1}	14	68	68
$\mathbf{p}_k$ (Ref. eq. 3)	12	26	39
β_k (Ref. eq. 5)	7	19	27
λ_k (Ref. eq. 5)	23	22	23
$\lambda_j \mathbf{p}_k^H, g_k$ (Ref. eq. 6)	7	12	14
Total	**81**	**180**	**221**
Symbol $\tilde{x}$ Throughput (Ref. eq. 6)	4 symbols/8 cycles	3 symbol/11 cycles	4 symbol/13 cycles
ASIC M Symbols/sec (@ 546 MHz)	273	149	168
FPGA M Symbols/sec (@ 130 MHz)	65	35.45	40

Table 3. EquASIP computation time for MMSE-IC$_1$ equations

of the resources are not used. This illustrates a typical tradeoff between flexibility, resource utilizations and system performance. The throughput for 2×2 Golden code is same as 4×4 SM.

4.3 Comparison with state of the art

In Table 4, different architectural parameters of state of the art implementations are summarized and compared with EquASIP implementation results. All of the referenced implementations present dedicated architecture for a specific system configuration except (Eilert et al., 2007) where the proposed architecture supports 2×2 and 4×4 matrix inversion. Table 4 is organized in such a way that first of all comparison is made with (Boher et al., 2008), (Kim et al., 2008) and (Karakolah et al., 2009) which provide a complete solution to generate estimated symbol vectors. Then comparison with (Myllyla et al., 2005) (providing solution to compute only the coefficient matrix of Equation 3) is tabulated. Finally, the EquASIP is compared with (Edman & Öwall, 2005), (Karkooti et al., 2005) and (Eilert et al., 2007) which provide architectures only for matrix inversion. Furthermore, in order to make a fair comparison, the EquASIP was synthesized with the same target technology as used in the implementation with which it is being compared.

The work presented in (Boher et al., 2008) is aimed at achieving fast fading 4×4 MIMO SM using MMSE-IC. This implementation uses $\sigma_{\hat{x}}^2 = 0$ in first iteration and $\sigma_{\hat{x}}^2 = \sigma_x^2$ in later iterations to simplify the architecture. However, while using in iterative context this assumption of perfect $\sigma_{\hat{x}}^2$ information induces a performance loss. Due to a fully pipelined architecture it outputs a vector containing four estimated symbols at every 38 clock cycle. Hence, the throughput is 1.31 Mega vectors at presented frequency. With EquASIP, working on same configuration, the cycles required for one symbol vector estimation are 234. This results in a throughput of 0.5 Mega vectors per second at considered frequency. Hence, the flexibility of EquASIP to support 5 different STC comes at the cost of 2.4 times less throughput, 53% more slice registers and 16 more dedicated multipliers compared to (Boher et al., 2008).

In (Kim et al., 2008), the architecture implements 4×4 MIMO SM detector for 802.11n standard. In this application the design is made for a worst case scenario where for 48 vectors channel remains constant. To decode a frame of 48 vectors, the work in (Kim et al., 2008) takes 388 clock cycles. Which results in 17.3 M vectors per second at a frequency of 140 MHz. When

Opr.	System Config.	Ref.	Algorithm	Target Device	Opr. Freq. (MHz)	Hardware Resources				Clock Cycle	Throughput (Mega Operations per sec)
						FPGA			ASIC		
						Slice/LE		HW	Area		
						Reg.	LUT	Mul.	(K Gates)		
MIMO Symbol Vector Estim.	4×4 SM Fast Fading	(Boher et al., 2008)	QR CORDIC	Startix	50	8670		12	-	38	1.31
		EquASIP	Analytical		120	13272		28	-	234	0.5
	4×4 SM Block Fading	(Kim et al., 2008)	QR CORDIC	Virtex-II	140	14166		103	-	388	17.31
		EquASIP	Analytical		83	8477		14	-	845	4.71
	2×2 PC Quasi Static	(Karakolah et al., 2009)	Blockwise Analytical	Virtex-V	60	817	2715	60	-	1	120
		EquASIP			130	3174	11299	14	-	3.25	40
$\mathbf{p}_k$ Eq.3	2×2 MIMO SM	(Myllyla et al., 2005)	QR CORDIC	Virtex-II	-	11910		20	-	685	-
			QR SGR		-	6305		59	-	415	-
		EquASIP	Analytical		83	8477		14	-	42	1.97
	4×4 MIMO SM	(Myllyla et al., 2005)	QR CORDIC		-	16805		44	-	3000	-
			QR SGR		-	-		-	-	-	-
		EquASIP	Analytical		83	8477		14	-	157	0.53
Matrix Inv.	4×4 Matrix	(Edman & Öwall, 2005)	QR SGR	Virtex-II	100	2224	2212	-	-	175	0.57
		EquASIP	Analytical		83	3177	15997	14	-	68	1.2
		(Karkooti et al., 2005)	QRD-RLS	Virtex-IV	115	9117		22	-	933	0.15
		(Eilert et al., 2007)	Blockwise Analytical	Virtex-IV	100	1716	2094	8	-	120	0.83
				90 nm	500	-	-	-	43	92	5.43
		EquASIP		Virtex-IV	117	3232	16091	14	-	68	1.7
				90 nm	546	-	-	-	85	68	8.02

Table 4. EquASIP performance comparison

comparing with our work, this EquASIP consumes 221 clock cycles to compute equalization coefficient for a frame and 13 clock cycles for each vector estimation. Hence the total consumed clock cycles for 48 vectors estimation are 221 + 13×48 = 845 which results in a throughput of 4.7 M vectors per second at a frequency of 83 MHz. Hence, throughput of the dedicated architecture of (Kim et al., 2008) is almost 3.6 times more at a cost of almost twice the FPGA slice used and 7.5 times more multipliers. Again this implementation is not flexible for variable antenna size, time selectivity of the channel and iterative nature of equalization.
The realization of 2×2 MMSE-IC equalizer in (Karakolah et al., 2009) includes pre-coding (PC). The equivalent channel matrix becomes a 4×4 matrix shown below:

$$\mathcal{H} = \begin{bmatrix} h_{11} & h_{12} & 0 & 0 \\ h_{21} & h_{22} & 0 & 0 \\ 0 & 0 & h_{11} & h_{12} \\ 0 & 0 & h_{21} & h_{22} \end{bmatrix}$$

The inversion of this matrix needs execution of two 2×2 matrix inversions. Other than this, to map this PC on the EquASIP, one 4×4 matrix multiplication is required to incorporate the PC matrix. The rest of the computations are same as required in 4×4 MIMO SM. Hence, to compute the equalization coefficients on EquASIP, 197 clock cycles will be consumed. For a target quasi-static environment, the EquASIP takes 197 cycles at 130 MHz as compared to the dedicated architecture taking 20 cycles at 61 MHz (Karakolah et al., 2009). This part is not crucial because it is computed once for a frame. The throughput of EquASIP is 40 Mega

symbols per second and hence 3 times less than the dedicated architecture. The 3 times faster output of dedicated architecture comes at 5 times multipliers used and this architecture used 4 times less slice registers and LUTs.
The EquASIP is better both in area and performance when compared with (Myllyla et al., 2005). While comparing with (Edman & Öwall, 2005), (Karkooti et al., 2005) and (Eilert et al., 2007), EquASIP outperforms these architectures in throughput. EquASIP occupies more area as compared to these dedicated implementations for matrix inversion as, besides its flexibility, EquASIP supports all functions required in MMSE-IC equalization algorithm.
In the above analysis, an attempt is made to compare dedicated and flexible architectures for MMSE-based equalization. In the presence of multiple system configurations and different variants of algorithms in the equalizer, EquASIP provides a promising flexible solution compared to dedicated implementations.

5. Conclusion

ASIP concept with associated efficient design and prototyping flows is emerging nowadays as a promising implementation solution for wireless communication applications. It is mainly the need to increase the flexibility and the opportunities for modular reuse that is pushing industry to use more and more software-programmable solutions for practically every class of digital components. Trade-offs between performance and flexibility can be tuned to the exact needs of the application, besides the already available design tools enable really efficient hardware synthesis and validation.
This chapter illustrates ASIP design and prototyping approach in wirless communication applications through a detailed example for MIMO detection. The first flexible ASIP implementing an MMSE-IC linear equalizer for turbo equalization application has been presented. Analysis and simulation of mathematical equations involved in MMSE-IC LE allowed to identify potential complex-numbered operations which lead to device the instruction set for the proposed EquASIP. The specific instructions for complex number arithmetic enable to efficiently perform computations on variable sized complex numbered matrices which in turn provide required flexibility in MMSE-IC and promote its reuse for other MMSE-based applications.
Flexibility of the presented EquASIP architecture allows its reuse for each of Alamouti code, Golden code, 2×2, 3×3 or 4×4 spatially multiplexed turbo MIMO application with BPSK, QPSK, 16-QAM, and 64-QAM. When targeting 90 *nm* technology, the proposed architecture enables a maximum throughput of 273 MSymbol/sec for 2×2, 148 MSymbol/sec for 3×3 and 168 MSymbol/sec for 4×4 MIMO systems. The presented original contribution demonstrates promising results using the ASIP approach to implement flexible, yet efficient, MMSE-based iterative MIMO equalizer.
Regarding research perspectives in this domain, the increasing low-power requirement should be considered as another optimizing design dimension to the ASIP solution for wireless communications applications. By inheriting the relevant low-power design techniques from already established low-power implementation schemes for programmable architectures, ASIP design flows can be optimized for power consumption. In the presented ASIP design example an attempt is made to provide maximum flexibility within the equalization application. This upper bound for flexibility implies similar bound for power consumption. Hence by examining the exact required flexibility for a particular system a trade-off can be achieved between flexibility and power demands.

6. Acknowledgment

This work was supported in part by UDEC and TEROPP projects of the French Research Agency (ANR).

7. References

Boher, L., Rabineau, R. & Hélard, M. (2008). Architecture and implmentation of an iterative receiver for MIMO systems, *International Symposium on Turbo Codes and Realted Topics*.

BOUVET, P.-J. (2005). *Récepteurs itératifs pour systèmes multi-antennes*, PhD thesis, INSA de Rennes France.

Burg, A., Borgmann, M., Wenk, M., Zellweger, M., Fichtner, W. & Boelcskei, H. (2005). VLSI Implementation of MIMO Detection Using the Sphere Decoding Algorithm, *IEEE Journal on Solid-State Circuits* 40(3).

Cavalec, K. A., Sicot, G. & Leroux, D. (2008). Reduced complexity near-optimal iterative receiver for Wimax full-rate space time code, *5th international symposium on Turbo Codes and related topics*.

Döhler, R. (1991). Squared Givens rotation, *IMA Journal of Numerical Analysis* 11(1): 1–5.

Edman, F. (2006). *Digital Hardware Aspects of Multiantenna Algorithms*, PhD thesis, Lund University, Department of Electroscience, Lund, Sweden.

Edman, F. & Öwall, V. (2005). A scalable pipelined complex valued matrix inversion architecture, *IEEE International Symposium on Circuits And Systems, ISCAS'05*, pp. 4489–4492.

Eilert, J., Wu, D. & Liu, D. (2007). Efficient complex matrix inversion for MIMO Software Defined Radio, *IEEE International Symposium on Circuits and Systems, ISCAS'07* .

Golub, G. H. & van Van Loan, C. F. (1996). *Matrix Computations (3rd Edition)*, The Johns Hopkins University Press.

Hoffmann, A., Schliebusch, O., Nohl, A., Braun, G., Wahlen, O. & Meyr, H. (2001). A Methodology for the Design of Application Specific Instruction set Processors (ASIP) Using the Machine Description Language LISA, *IEEE/ACM International Conference on Computer Aided Design, ICCAD'2001.*, pp. 625–630.

Jafri, A. R., Baghdadi, A. & Jezequel, M. (2009). ASIP-Based Universal Demapper for Multiwireless Standards, *IEEE Embedded Systems Letters* 1(1): 9–13.

Jafri, A.-R., Karakolah, D., Baghdadi, A. & Jezequel, M. (2009). ASIP-based Flexible MMSE-IC Linear Equalizer for MIMO Turbo-Equalization Applications, *IEEE/ACM Design, Automation and Test in Europe Conference & Exhibition, DATE'09*.

Karakolah, D., Jégo, C., Langlais, C. & Jézéquel, M. (2009). Design of an iterative receiver for linearly precoded MIMO systems, *IEEE International Symposium on Circuits and Systems, ISCAS'09* .

Karkooti, M., Cavallaro, J. R. & Dick, C. (2005). FPGA Implementation of Matrix Inversion Using QRD-RLS Algorithm, *Asilomar Conference on Signals, Systems and Computers*, pp. 1625–1629.

Kim, H., Zhu, W., Bhatia, J., Mohammad, K., Shah, A. & Danesrad, B. (2008). A Practical Hardware Friendly MMSE Detector for MIMO-ODFM-Based Systems, *EURASIP Journal on Advances in Signal Processing* .

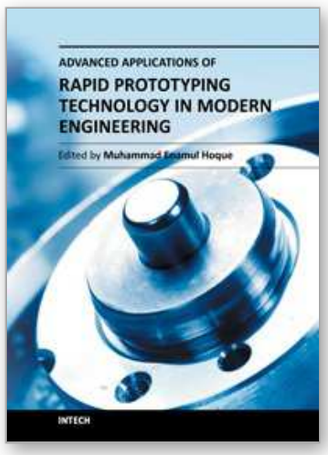
Laot, C., Le Bidan, R. & Leroux, D. (2005). Low-complexity MMSE turbo equalization: a possible solution for EDGE, *IEEE Transactions on Wireless Communications* 4(3): 965–974.

Mitola, J. (1995). The Software Radio Architecture, *IEEE Communications Magazine* 33(5): 26 –38.

Muller, O., Baghdadi, A. & Jezequel, M. (2009). From Parallelism Levels to a Multi-ASIP Architecture for Turbo Decoding, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 17(1): 92–102.

Myllyla, M., Hintikka, J.-H., Cavallaro, J., Juntti, M., Limingoja, M. & Byman, A. (2005). Complexity Analysis of MMSE Detector Architectures for MIMO OFDM Systems, *Asilomar Conference on Signals, Systems and Computers*, pp. 75 –81.

Volder, J. E. (1959). The Cordic Trigonometric Computing Technique, *IEEE Transactions on Electronic Computers* 8(3): 330–334.



Advanced Applications of Rapid Prototyping Technology in Modern Engineering

Edited by Dr. M. Hoque

ISBN 978-953-307-698-0
Hard cover, 364 pages
Publisher InTech
Published online 22, September, 2011
Published in print edition September, 2011

Rapid prototyping (RP) technology has been widely known and appreciated due to its flexible and customized manufacturing capabilities. The widely studied RP techniques include stereolithography apparatus (SLA), selective laser sintering (SLS), three-dimensional printing (3DP), fused deposition modeling (FDM), 3D plotting, solid ground curing (SGC), multiphase jet solidification (MJS), laminated object manufacturing (LOM). Different techniques are associated with different materials and/or processing principles and thus are devoted to specific applications. RP technology has no longer been only for prototype building rather has been extended for real industrial manufacturing solutions. Today, the RP technology has contributed to almost all engineering areas that include mechanical, materials, industrial, aerospace, electrical and most recently biomedical engineering. This book aims to present the advanced development of RP technologies in various engineering areas as the solutions to the real world engineering problems.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Atif Raza Jafri, Amer Baghdadi and Michel Jezequel (2011). ASIP Design and Prototyping for Wireless Communication Applications, Advanced Applications of Rapid Prototyping Technology in Modern Engineering, Dr. M. Hoque (Ed.), ISBN: 978-953-307-698-0, InTech, Available from: http://www.intechopen.com/books/advanced-applications-of-rapid-prototyping-technology-in-modern-engineering/asip-design-and-prototyping-for-wireless-communication-applications

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen