ASIP Architecture for Multi-Standard Wireless Terminals

D. Lo Iacono, J. Zory, E. Messina, N. Piazzese, G. Saia, A. Bettinelli
Advanced System Technology
STMicroelectronics
daniele.loiacono@st.com

Abstract

This paper presents the Block Processing Engine (BPE), an Application Specific Instruction-Set Processor (ASIP) explicitly designed for the implementation of multi-standard wireless terminals. Thanks to a high level of parallelism and a consistent use of pipeline, the BPE architecture fully satisfies stringent real-time constraints imposed by emerging technologies. Its efficiency has been proven through the implementation, the physical synthesis for the CMOS 90nm STM technology and the FPGA prototyping on the ARM Versatile platform of a dual-standard Frequency Domain Equalizer (FDE) supporting the 3GPP HSDPA and the IEEE 802.11a standards.

1. Introduction

Wireless communications are rapidly evolving toward a large variety of systems offering high data-rate multimedia services. Next generation terminals must be more and more powerful and still flexible enough to support rapid modifications of emerging standards or even multiple standards. Architectures devoted to base-band processing must endure high computational efforts while exhibiting the potential of switching between different algorithms or systems on-demand. In this context, traditional approaches such as ASIC and DSP reveal some limitations. Indeed ASIC design is highly efficient in terms of computational capacity, power consumption and real-time processing capabilities. However its lifetime is strongly shortened by the limited flexibility, and is definitely inadequate to stay on the track of a continuously changing market. Recent DSPs, although powerful and flexible, are still not capable of sustaining the data-rates imposed by most of the emerging systems, whose requirements are growing even faster than the DSPs technology itself [1][2]. Despite of its limitations, ASIC design is often considered as the only way to achieve high-throughputs. In this scenario, emerging Application-Specific Instruction-set Processors (ASIPs) seem an attractive way of satisfying real-time requirements while assuring a certain degree of flexibility. Exempt from supporting general-purpose applications, ASIPs make use of a reduced set of dedicated instructions with high level of parallelism, hence allowing efficiencies comparable to that of ASICs [2].

This paper presents the Block Processing Engine (BPE), a platform embedding a programmable processor capable of simultaneously addressing a set of customized hardware modules specifically suited for wireless processing. To prove the potential of the BPE, a dual-standard Frequency Domain Equalizer (FDE) supporting WCDMA/HSDPA and OFDM/802.11a has been implemented, synthesized and validated on FPGA.

2. Architecture description

2.1. General overview

The main feature of the Block Processing Engine (BPE) is the support of an extended instruction-set acting on a collection of Dedicated Processing Units (D-PU).

The BPE system architecture is shown in Figure 1.

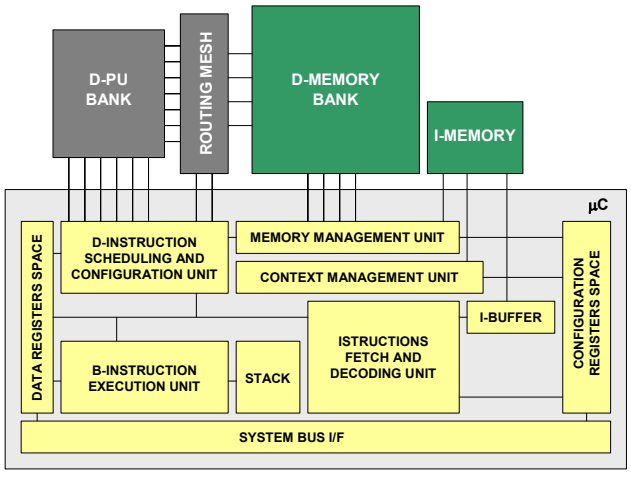

Figure 1. Block Processing Engine architecture.

The core function is a programmable controlling unit (µC) supporting Basic Instructions (*B-instruction*) mainly

devoted to the flow control, and Dedicated Instructions (*D-instruction*) acting on the *D-PU bank* embedding a collection of D-PU performing intensive signal processing on complex streams of data.

Starting from a program stored into the *I-memory*, the instructions are fetched, decoded, and properly allocated according to their type. More precisely, B-instructions are dispatched to the μC execution unit for local execution, while D-instructions are scheduled on the D-PU bank.

To match the stringent real-time constraints imposed by emerging high-throughput systems, the μC unit has been conceived to support parallel and pipelined processing on complex arrays of data. The capability of treating arrays instead of scalars reduces the latency overhead due to the execution of multiple scalar operations on groups of data, but requires a memory sub-system consisting of a bank of concurrently accessible SRAM (D-memory bank).

High levels of parallelism can be achieved supporting the concurrent execution of D-instruction bundles, while pipelined processing requires the capability of directly connecting a set of D-PUs to form a processing chain. To fully support parallel and pipelined processing, the BPE has been equipped with a *routing mesh* enabling all the point-to-point connections between the D-PUs as well as any possible connection between the D-PU bank and the D-memory bank. A schematic view of the routing mesh is depicted in Figure 2, where is visible the output routing unit supporting data buffering for D-PUs synchronization.

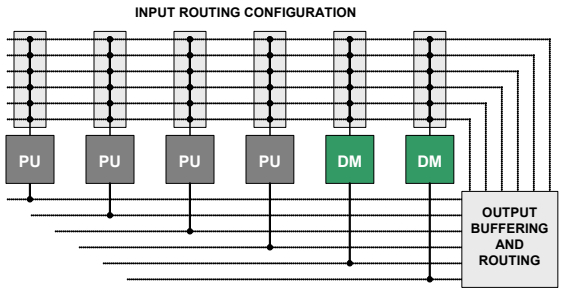

Figure 2. Routing mesh structure.

To support the implementation of multi-mode or multi-standard terminals, the μC embeds a context switching unit capable of saving the status of the program under execution and dynamically switching to another program stored into a different logical partition of the I-memory.

The μC is programmed through the *System Bus I/F* by a host entity called *Communication Master* (CM). Drivers accessing the μC act mainly on the *configuration registers space*, and can be implemented either in C or using the CM native assembly language. Using the drivers, the CM can perform a set of requests such as loading/saving data from/to any data memory of the D-memory bank, loading programs into the I-memory, running the execution of a program. Moreover, the CM can access the μC registers of the *data registers space* to update program parameters at run-time, to retrieve feedback values from the D-PUs or to monitor the status of the D-PU bank during the execution of the program.

2.2. D-PUs structure and D-instructions

As stated, the μC is in charge of scheduling the D-PU bank activity on the basis of a program loaded by the CM into the I-memory. The D-PU bank must be intended as a collection of objects (with a given granularity) each embedding a set of functional units performing specific processing on complex data arrays. Since parallelism can be activated at D-PU level only, D-instructions addressing the functional units of the same D-PU can not be executed concurrently. They are considered by the μC as part of the D-PU instructions sub-set. It can be argued that the trade-off between processing speed and flexibility strongly depends on the granularity of the D-PU set as well as on the adequate splitting of potentially concurrent functions within the D-PU bank. The granularity choice is crucial since it is responsible of the D-PUs re-use degree.

The basic structure of a D-PU is depicted in Figure 3.

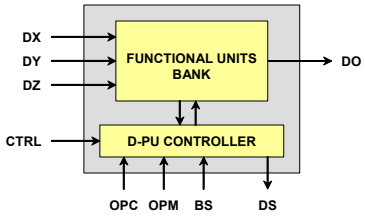

Figure 3. D-PU basic structure.

Each functional unit within the D-PU is addressed by the following assembly-like instruction:

```
PUN.OPC[.OPM] [BS] [DX[,DY,DZ]] DO          (1)
```

where PUN indicates the D-PU number, OPC indicates the operation code addressing a specific D-PU functional unit, OPM indicates the operation mode for the specific OPC, BS is the array (block) size, while DX, DY, DZ and D0 are the μC registers holding the information needed to retrieve the three input operands and the output operand respectively. Instruction operands can be either real or complex scalar (registers file) or array (D-memory bank).

For instance, the D-instruction below uses the ALU to perform the complex multiplication (MUL) between an array of size B0 stored into the D-memory M0 and a scalar stored into the μC data register R1:

```
ALU.MUL.RND B0 M0,R1 M1
```

The output vector, rounded over 16-bit according to the specified OPM (RND) is stored into the memory M1.

After the D-instruction execution, each D-PU updates the μC D-status registers (DS). Commonly these registers are used to count the overflow occurrences during array processing, and can be useful to detect for instance quantization problems within the processing chain.

The mnemonic format (1) corresponds to a fixed-length 32-bit operating code supporting up to 16 D-PU equipped with 8 functional units each having 4 operating modes.

2.3. Pipelined processing using macros

On top of classic concurrent processing, the BPE allows the interconnection of D-PU sub-sets implementing more complex functions, otherwise called *macros*. A macro is a way of pipelining the processing among the D-PU without having to perform intermediate accesses to the D-Memory bank. This has consistent advantages in terms of speed and power consumption. To associate a D-PU sub-set with a macro, each D-PU must be explicitly linked using a special μC register (LX).

For instance, the following macro adds the complex value R0 to all the elements of the complex conjugate of the FFT of the array residing into the memory M0:

```
M1 = ADD(CNJ(FFT(M0)),R0)                    (2)
```

It can be implemented linking the D-PU as follows:

```
FTU.FFT B0 M0 LX
CLU.CNJ B0 LX LX                             (3)
ALU.ADD.SAT B0 LX,R0 M1
```

Figure 4 shows the latency reduction when using the macro (3).

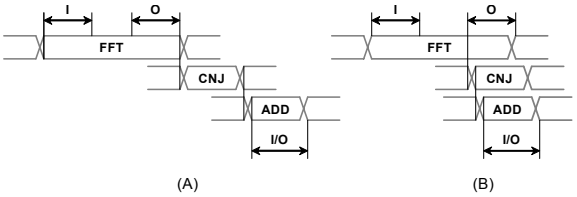

Figure 4. Normal (A) and linked (B) executions.

It must be noted that since intermediate results of a macro are directly propagated from D-PU to D-PU, they will not be anymore available after the macro execution.

2.4. Parallel processing

To perform concurrent processing, the program flow is broken into bundles of instructions that are executed at once. Each bundle can be formed by either B-instruction only (B-bundle) or D-instruction only (D-bundle).

Since there is no parallelism within the B-instructions set, a B-bundle merely indicates the chance of pipelining the execution of a group of B-instruction when there is no direct dependency among them, as depicted in Figure 5.

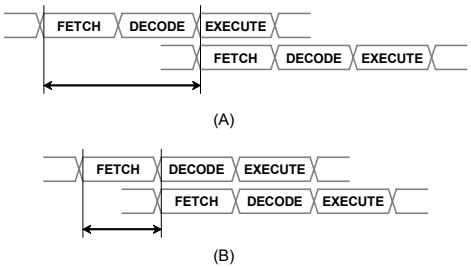

Figure 5. Normal (A) and B-bundle (B) execution.

Unlike the B-bundle, the D-bundle fully exploits the parallelism within the D-PU bank. All the D-instruction within a D-bundle are executed concurrently. Moreover, the μC *scheduling and configuration unit* support macros within the D-bundle, thus allowing both parallel and pipelined processing within the same D-bundle.

The following example shows a D-bundle embedding the macro (3):

```
CGU.SCR.INI B0 D1,D2 ;
FTU.FFT B0 M0 LX
CLU.CNJ B0 LX LX                             (4)
ALU.ADD.SAT B0 LX,E1 M1 ;
HTU.FHT B1 M2 M3
```

It can be noted that concurrency is explicitly indicated using the semicolon operator, as for VLIW processors [1].

Figure 6 shows the execution of the D-bundle (4), where are clearly visible the D-PU configuration (CFG) and the execution (EXE) phases.

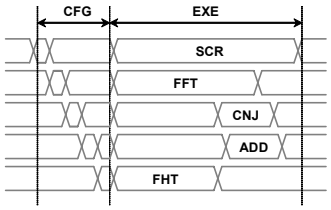

Figure 6. D-bundle configuration and execution.

During the configuration phase, the μC sets the routing mesh, provides activation signals to the involved D-PUs and schedules all the D-memory bank accesses. Execution phase is then activated simultaneously on all the D-PUs within the D-bundle.

2.5. Considerations on complexity and speed

Compared to an ASIC implementation, the BPE suffers the major drawback of inherently introducing an overhead in terms of both complexity and speed. The impact of the additional logic, especially the μC and the routing mesh, on the overall design needs to be carefully investigated. It can be shown that the μC complexity depends linearly on the number of D-PUs that have to be simultaneously activated (i.e. the D-bundle size), while the dependency of the routing mesh from the D-PU bank size is quadratic.

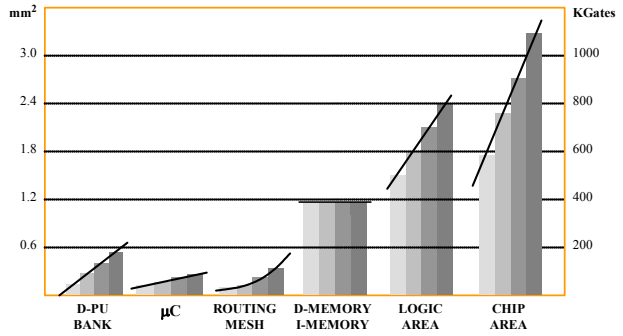

Figure 7. Complexity as a function of D-PU bank size.

Physical synthesis for the 90nm CMOS STM low-power technology (1.20V) has been performed using Synopsys Physical compiler. Results of Figure 7 refer to an implementation embedding a D-Memory bank of 4 SRAM of 4K×32 bit each, a I-memory of 4K×32 bit, and a D-PU bank composed by an increasing number of *dummy* D-PU (small ALU composed by a complex multiplier and an adder) ranging from 4 to 16 (maximum number of D-PU allowed by the 32-bit D-instruction operation code). The logic area includes the logical cells only, while the chip area takes into account the layout constraints and the net routing. Gate equivalent complexity can be roughly evaluated using a density of 330 KGates/mm^2 for the core logic and a density of about 1.6 mm^2/Mbits for the SRAM.

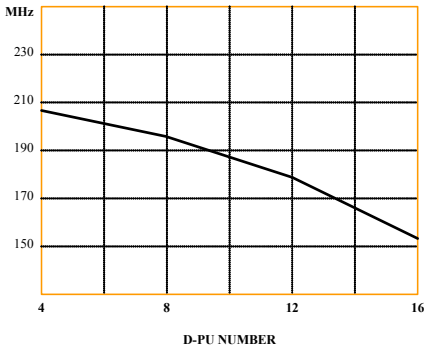

Figure 8. μC speed as a function of the D-PU number.

In terms of processing speed, and finally of achievable throughput, the μC core frequency is fundamental, and must be sufficiently high to avoid introducing a bottleneck when embedding fast D-PUs. Synthesis results of Figure 8 show a smooth dependency of the μC frequency from the number of D-PU, and a still reasonable worst value.

3. Dual-standard HSDPA/WLAN equalizer

3.1. System description

As a case study, the BPE has been used to implement a re-configurable equalizer supporting OFDM and CDMA, and more specifically capable of demodulating WLAN 802.11a as well as multi-code UMTS/HSDPA signals. In order to fully exploit the re-usability aspects of the BPE, the equalization has been performed into the frequency domain. While frequency domain equalization (FDE) is commonly employed for multi-carrier systems, its use on single-carrier systems has been investigated only in the last few years [3][4]. Recent studies have demonstrated that FDE for WCDMA is not only feasible, but even less complex than the time-domain approach [5][6][7].

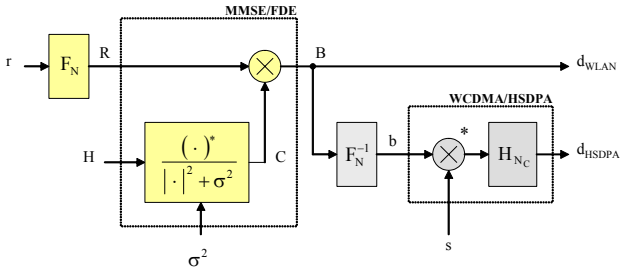

Figure 9. Frequency Domain Equalizer (FDE).

The block diagram of Figure 9 gives a clear picture of what can be re-used within the dual-mode HSDPA/WLAN equalizer. The pure MMSE/FDE equalization is common to both systems, while the WCDMA/HSDPA multi-code demodulation needs some additional units strictly related to CDMA systems, such as the scrambling code and the de-spreader (here implemented through Walsh-Hadamard transform to support multi-code despreading).

The MMSE/FDE is performed using the following element-wise equation:

$$B_i = \frac{H_i^*}{|H_i|^2 + \sigma^2} \cdot R_i \,, \qquad (5)$$

where **R**, **H** and **B** are the Fourier transforms of the received signal, the channel impulse response and the estimated transmitted signal respectively, while σ^2 represents the noise variance.

The channel response **h**, which in Figure 9 is assumed already available into the frequency domain (**H**), strictly depends on the system. For OFDM systems the channel is estimated correlating the frequency domain signal **R** with the local replica of the frequency domain preamble. For WCDMA, the channel is evaluated into the time-domain using the available pilot channel and then transformed into the frequency domain for subsequent FDE.

Although MMSE/FDE is common to both systems, it will be shown that the actual implementation requires different FFT sizes. In OFDM systems the FFT size corresponds to the number of sub-carriers, and thus is fixed by the standard (64-point for the IEEE 802.11a). For CDMA, the FFT size can be considered as a performance parameter depending on the channel delay spread. In fact, the equalization of adjacent blocks requires applying overlap-and-save in order to reduce the boundary errors due to the absence of cyclic prefix [3][4]. Since the overlap-and-save factor is fixed by the delay spread, the percentage of useful symbols per block, and ultimately the throughput, can be adjusted acting on the FFT size only. A good trade-off between throughput and complexity is using a 256-point FFT for an overlap-and-save factor of 16-chip [7]. HSDPA multi-code de-spreading is then performed by first descrambling the time-domain equalized signal with the cell-specific code, and then evaluating the N_C-point Fast Hadamard Transform (FHT) of the descrambled signal to simultaneously demodulate all the codes associated with the spreading factor N_C [7].

3.2. Equalizer implementation

The BPE implementation of the dual-mode equalizer passes through the identification of the set of D-PU needed to perform the processing of Figure 9. For an optimal resources re-use, all the functional units within the D-PU bank must be primarily divided into common functions and specific functions. Typically, common functions with fine granularity can be embedded into one or more ALU.

The key D-PU is the FTU (Fourier Transform Unit), which must support dynamic size scaling and FFT/IFFT switching on-demand. The FTU has been implemented as a single Radix-4 butterfly, serially re-used to perform the overall processing. To perform each butterfly calculation in one clock cycle, the FTU has been equipped with two ping-pong SRAM banks working alternately for even and odd stages, while the coefficients ROM has been divided into banks providing all the twiddle factors at once [7].

Moreover, the D-PU bank has been equipped with a specific D-PU generating the UMTS scrambling code. The Code Generation Unit (CGU) embeds a unique functional unit supporting different OPM. The INI operation mode initializes the generator storing a certain number of seeds corresponding to different delays (within the code period) into the generator internal memory, thus allowing fast initialization when delayed replicas of the code have to be generated. The RUN operation mode generates the binary code on the basis of the code number and the delay value provided as input operands. The PCK (pack) operating modes still generates the code, but allows packing shifted replicas of the code into 16-bit complex word to speed-up the subsequent correlations. In fact, the CGU is designed to work with the Correlators Bank Unit (CBU), which embeds 16 accumulators providing up to 16 correlation values for each clock cycle.

Multi-code HSDPA demodulation is performed using the Hadamard Transform Unit (HTU) embedding the FHT, whose implementation is based on classical Radix-2 algorithm.

The rest of the operations, such as MMSE calculation, is performed using two ALUs supporting complex arrays arithmetic. It is worth noting that the coexistence of two ALUs is crucial to minimize the latency. In fact, since parallelism is at D-PU level only, concurrency between arithmetic operations or even the use of macros would not be possible using a single ALU.

As stated, each D-PU corresponds to a D-instructions subset contributing to the complete D-instructions set of the BPE. Thus, the D-instructions set coming from the already described D-PU bank has been used to write the software efficiently implementing the dual-mode FDE.

To add flexibility and to allow quickly testing different scenarios, the most relevant system parameters are mapped into the μC data registers space, so that they can be updated at run-time by the CM. These registers hold dynamic parameters passed by upper layers or by different functional blocks of the physical layer, but they can be reserved also to support future modifications or evolution of the standard. For instance the scrambling code number, the noise variance, the number of DATA fields in a WLAN frame, but also the HSDPA spreading factor, the WLAN pilot carriers number and even the FFT size have been considered as parameters passed to the program at execution time.

3.3. Performance evaluation

The RTL implementation of the HSDPA/WLAN equalizer has been validated against test vectors generated from a bit-true SystemC reference model. Anticipating the integration of the WLAN support into UMTS/HSDPA mobile terminals, the equalizer has been tested using the ITU Pedestrian B multi-path channel model, as specified by the 3GPP performance requirements for HSDPA [8].

Figure 10 shows the BER curves of the dual-mode FDE when using un-coded QPSK modulation for both WLAN and HSDPA. The WLAN frame holds 23 DATA fields, while the HSDPA uses a spreading factor of $N_C = 16$.

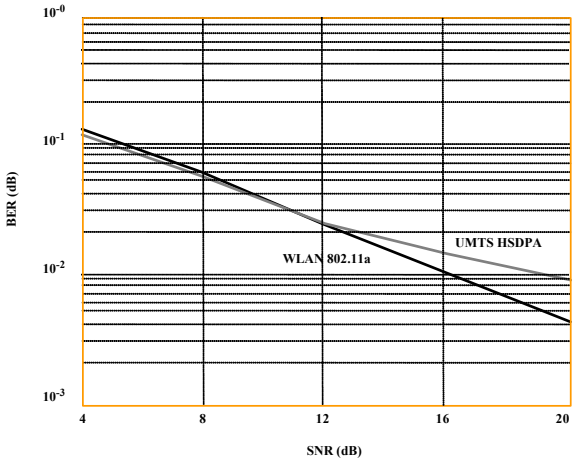

Figure 10. Performance of the dual-mode equalizer.

3.4. Real-time processing

As discussed, real-time processing capabilities depend on the way the D-instructions are scheduled on the D-PU bank, i.e. on the software implementation. The timing diagram of Figure 11 shows the D-PUs activity during the equalization of a WLAN frame. The large use of macros is fully justified by the pipelined nature of the base-band processing. As expected, most of the processing latency is due to the FFT calculation. Since each D-PU within a macro is allocated during the configuration phase, the FFT forces the ALU to stall until the output is available. The overall latency is reduced by the joint use of two ALU, properly linked in a macro to perform channel estimation averaging and MMSE coefficients calculation.

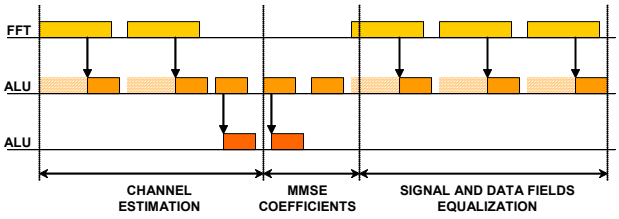

Figure 11. D-PUs scheduling for WLAN equalization.

The latency to demodulate a QPSK HSDPA un-coded slot, made of channel estimate, FDE, descrambling and multi-code de-spreading is about 36K clock cycles. When using a clock speed of 200MHz, the entire demodulation lasts for about 180μs (i.e. about 27% of the entire slot duration). Similarly, the latency of an un-coded QPSK IEEE 802.11a frame with 23 DATA fields takes about 5K cycles, corresponding to 25μs (about 30% of the frame duration). These results confirm the expectation of real-time running both systems.

3.5. FPGA prototyping

The overall ASIP design comprehensive of the D-PU bank for UMTS/WLAN FDE has been mapped on FPGA for fast prototyping. The system-on-chip environment employing the BPE as slave peripheral has been emulated on the *ARM™ Versatile* platform providing the *Xilinx XC2V8000* FPGA on which the BPE has been uploaded. The on-board ARM9-based development chip has been used as CM running the C-language drivers for BPE interfacing using the AMBA/AHB bus protocol.

4. Conclusions

It has been presented the Block Processing Engine (BPE), a programmable architecture embedding dedicated instructions acting on a set of processing units specifically suited for intensive base-band processing. To prove its efficiency, the BPE has been employed to implement a dual-mode HSDPA/WLAN frequency-domain equalizer. Synthesis results and latency evaluation have confirmed the expectation of real-time running both systems. The high degree of flexibility and the consistent use of parallel and pipelined processing makes the BPE an attractive solution for the implementation of high-throughput next generation multi-standard wireless terminals.

References

[1] J. Eyre, J. Bier, “The Evolution of DSP Processors”, IEEE Signal Processing Magazine, Mar. 2000.

[2] J. R. Cavallaro, P. Radosavljevic, “ASIP Architecture for Future Wireless Systems: Flexibility and Customization”, 11th Wireless World Research Forum, Oslo, Jun. 2004.

[3] D.D. Falconer, S.L. Ariyavisitakul, A.B. Seeyar, B. Edison, “Frequency Domain Equalization for Single-Carrier Broadband Wireless Systems”, IEEE Comm. Magazine, vol. 40(4), pp. 58-66, Apr. 2002.

[4] D.D. Falconer, S.L. Ariyavisitakul, “Broadband wireless using single carrier and frequency domain equalization”, IEEE 5th Symposium on Wireless Personal Multimedia Comm., vol. 1, pp. 27-36, Oct. 2002.

[5] I. Martoyo, T. Weiss, F. Capar, F.K. Jondral, “Low Complexity CDMA downlink receiver based on frequency domain equalization”, IEEE 58th Vehicular Technology Conference, vol. 2(6-9), pp. 987-991, Oct. 2003.

[6] J. Pan, P. De, A. Zeira, “Low Complexity Data Detection using Fast Fourier Transform Decomposition of Channel Correlation Matrix”, IEEE Global Telecom. Conference, vol. 2, pp. 1322-1326, Nov. 2001.

[7] D. Lo Iacono, E. Messina, C. Volpe, A. Spalvieri, "Serial Block Processing for Multi-Code WCDMA Frequency Domain Equalization”, IEEE Wireless Communications and Networking Conference, New Orleans, Mar. 2005.

[8] 3GPP TS 25.101, “User Equipment (UE) Radio Transmission and Reception (FDD)”, *Technical Specification Group RAN, 3GPP.*

[9] IEEE Std 802.11a-1999 – Part11 “W-LAN MAC and PHY layer specifications, *IEEE*.