Chapter 9

MATLAB as a Design and Verification Tool for the Hardware Prototyping of Wireless Communication Systems

Oriol Font-Bach, Antonio Pascual-Iserte, Nikolaos Bartzoudis and David López Bueno

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/48706

1. Introduction

The ability to verify the capacity gains of novel signal processing techniques or the performance of new communication standards is one of the main research and development drivers of both academic and industrial entities. In this context, the signal processing community has adopted MATLAB as a flexible modelling, simulating and testing software development environment. MATLAB includes numerous toolboxes, open-source code and pre-compiled libraries, which facilitate the design of complex systems using high-level models and provides the means for rapid verification of signal processing algorithms and systems in a user-controlled environment. The growing number of its add-on features allows MATLAB to fill the gap between these high-level models and the physical implementation of systems; e.g., a real-time Field Programmable Array (FPGA)-based prototype. Moreover, the functionality of MATLAB is significantly extended with the use of Simulink [1], which serves as a schematic-entry design and programming environment. The integration of the System Generator blockset of Xilinx [2] to Simulink and the direct linking of the latter with the Xilinx FPGA-design toolchain enriches the target use-cases of the software. This approach allows the creation of FPGA binary executables from high-level models. MATLAB is also one of the most popular software-modelling environments, whose functionality is commonly interfaced nowadays with instruments to provide connectivity, control and programming solutions for rapid prototyping and testing. In fact, MATLAB scripts are increasingly used to program a wide variety of testing, signal generation and signal analysis hardware instruments. Thus, the programming versatility of MATLAB allows it to be used as a key software component in complex testbeds, which comprise a multitude of software programming interfaces and heterogeneous hardware instruments. The role of such testbeds is crucial because they enable the prototyping and validation of advanced research concepts under realistic conditions,

providing at the same time a detailed account of hardware requirements and implementation feasibility.

The present chapter aims at describing how MATLAB can be used as a design and verification tool in the different phases of migrating a high-level model to a real-time hardware prototype, using as a case study the implementation of a real-life wireless communication system. The chapter proposes a generic design methodology and, finally, provides a practical case study related to the implementation of a real-time Multiple Input Multiple Output (MIMO) mobile WiMAX (i.e., IEEE 802.16e) system [3, 4].

1.1. Considered development scenarios

Real-time system-prototyping using FPGA devices is a painstaking and time-consuming process that goes beyond a controlled computer simulation. In this context, MATLAB is having a manifold contribution as a design and validation tool. In order to successfully leverage the advices, techniques and design methodology, it is required to define the specific development scenarios that have to be considered by digital design developers.

It is important to note that this chapter will not cover model-based, MATLAB-to-Register Transfer Level (RTL) design flows (e.g., by using the Simulink and System Generator tools). Adversely, a custom-code programming strategy will be followed, where the user carefully designs each component of the system and takes into account the constraints of real-world hardware and signals. Our focus is to unveil the key role that MATLAB plays when the design objective is the creation of custom Hardware Description Language (HDL) code (e.g., Very High Speed Integrated Circuit - VHSIC - HDL, VHDL) that targets high-performance wireless communication prototypes. In fact, converting a MATLAB model into a working VHDL code for such FPGA-based prototypes requires a considerable effort. Although the automatic MATLAB-to-HDL conversion is becoming increasingly popular, its efficiency is still under scrutiny by the FPGA designer community [5]. The main concern raised is that the MATLAB-to-HDL automatic conversion is not yet mature enough to cover the needs of processing demanding FPGA-based systems, where performance and constraints imposed by the size and the embedded resources of the target device, may occasionally render this option unsuitable. The direct MATLAB-to-HDL translation accepts only very limited constructs that can be automatically translated into hardware [6]. Other approaches, involving an intermediate stage of Matlab-to-C code generation, can be used as an alternative. The produced C code is consequently processed by C-to-HDL synthesis tools subject to certain modifications (i.e., the generated C code contains unsupported constructs that prevent a seamless translation to HDL code).

As already mentioned, the automatically-produced HDL code is usually not as efficient as the custom hand-written HDL one. This difference is becoming a significant factor to be considered when stringent FPGA area utilization conditions apply or when performance and achievable clock frequencies do matter [7]. The modern FPGA devices and the corresponding synthesis tools seem to address the issues mentioned before. This is due to the extraordinary capacities of the new devices in terms of embedded resources (logic, memories dedicated Digital Signal Processing - DSP - logic) and the significant improvement of the FPGA design and implementation tools. However, it is anticipated that the FPGA-based prototyping and the respective FPGA design tools are due to be challenged soon by the constantly aggregated performance requirements and algorithmic complexity of next generation wireless

communication systems. Therefore, an incremental design approach based on custom-HDL coding is once again expected to be the most reliable solution to sort out well-established digital design problems (i.e., dense FPGA designs with compute intensive requirements and hard to achieve timing constraints [8]). The only difference is that the complexity of such problems is scaled because of the massive amount of FPGA logic, memories and embedded components that need to be addressed. Custom HDL coding provides the means to control every important aspect of the design, which requires an in-depth knowledge of the low-level RTL architecture.

The design and validation principles presented herein could be applied in many digital-design cases. Nonetheless, the application-domain will be narrowed down to well-characterized case studies, in order to help the reader to assimilate the described concepts, methodology and examples. Thus, this chapter explores the uses of MATLAB when the custom-HDL design flow is employed for the prototyping of systems with design and implementation requirements similar to the ones described next:

I Real-time system prototyping

- *Advanced wireless communication system*: Algorithms and hardware technologies able to offer data rates higher than current systems are needed to cope with the requirements of emerging wireless communication systems. The MIMO technology, using multiple antennas both at the transmitter and receiver sides, combined with Orthogonal Frequency Division Multiplexing (OFDM) constitute a suitable technique for the implementation of advanced wireless communication systems. Additionally, the Orthogonal Frequency Division Multiple Access (OFDMA) is used to target Multi-User (MU) scenarios in high mobility conditions. A prominent MIMO configuration scheme proposed in OFDMA systems is the closed-loop one, where the receiver is providing information to the transmitter related to the current channel conditions by means of a dedicated feedback link. This improves the performance and usage of resources in scenarios with multiple competing users and fast channel fading (e.g., it is applied adaptive carrier allocation, Adaptive Modulation and Coding - AMC). The scenario can be augmented by contemplating an adaptive power-aware PHY-layer that takes into account the interaction with higher layers of the communication stack and user requirements (e.g., in terms of quality of service, monetary cost or battery constraints). The compliance with a modern wireless communication standard (e.g., mobile WiMAX, Long Term Evolution - LTE) also adds strict design requirements.
- *Real-time operation*: The real-time operation implies transmission and reception of an uninterrupted data flow. To tackle the challenges of real-time operation, especially when accounting for wide bandwidth at baseband, a low-latency pipelined processing structure has to be designed. The latter requires a large amount of memory resources for the intermediate data storage and implies a complex control plane, which usually features multiple clock domains. Moreover, the operation of high performance wireless communication systems results in a growth of the design, implementation and validation complexity. Notwithstanding, the real-time operation gives the opportunity to realize closed-loop strategies requiring dynamic adaptation of the system in response to the actual channel conditions.

- *FPGA-based prototyping*: The inherent parallelism of FPGA devices is providing the means to prototype bit-intensive systems following an RTL-design approach. In this context, the designer has to evaluate the computational, storage and timing requirements of the target FPGA-based platform, in order to ensure that the implementation is feasible. Additionally, the FPGA-based prototyping of baseband DSP algorithms using a custom HDL design flow, typically implies the use of fixed-point logic. Therefore, an optimum trade-off between the implementation complexity and the precision of the internal calculations has to be defined (i.e., maximizing the dynamic range at baseband). The effort of interfacing the user design with the on-board buses, peripherals and components residing outside the FPGA device (e.g., Analog-to-Digital - ADC - and Digital-to-Analog - DAC - circuitry) is a critical part of the on-board validation, because it can be proved quite costly in terms of time. Finally, the losses introduced by the ADCs, DACs and baseband digital logic can be calculated to quantify the precision of the FPGA-based prototype.
- *Heterogeneous hardware setup*: The validation of high performance FPGA-based prototypes requires close to real-world testing conditions, which provide the means to properly tune the operating behaviour according to the defined deployment-scenarios. This in turn implies the use of a testbed which features a heterogeneous hardware setup. A real-time testbed typically comprises Radio-Frequency (RF) front-ends, signal generation and signal acquisition hardware boards, FPGA-DSP based baseband boards and other specialized equipment (e.g., radio channel emulator, digital oscilloscope). Moreover, testbeds have data-capturing interfaces that enable the performance characterization of the system (i.e., off-line data post-processing and metric calculation in MATLAB-space).

II Offline system prototyping

- *Advanced wireless communication system*: The goal in this case is the rapid prototyping of advanced techniques able to satisfy the requirements of future wireless communication systems. As it will be detailed in the following lines, the prototypes that principally operate offline make a series of assumptions to simplify the testing and deployment conditions and remove or ignore real-life implementation constraints. This inevitably results in a partial validation of the systems under test, especially for those cases where exhaustive offline data processing is practically impossible. Nonetheless, their contribution is also significant because they enable the design and preliminary experimental evaluation of algorithms beyond the state-of-the-art.
- *Off-line operation*: One of the main drivers of rapid prototyping is based on hybrid experimental testbeds that combine real-time processing and offline software-based post-processing. Such platforms, make use of commercial Vector Signal Generator (VSG) instruments equipped with arbitrary waveform generators. These are configured with user-generated MATLAB vectors, which represent the output of a baseband transmitter and eventually produce a real-time RF signal that is transmitted using either antennas or a direct cable connection. Offline testbeds may also use a RF channel emulator or other instruments that combine signal generation and channel fading. On the receiver side the data is stored in large buffers (e.g., FPGA) and retrieved in order to be post-processed offline. The captured signals are used as test vectors that facilitate the modelling of the baseband signal processing algorithms of the receiver (i.e., MATLAB high-level model of the system). This prototyping methodology allows the rapid verification of the functionality and performance of

algorithms. However, certain data capturing and post-processing limitations apply, especially when the testing requires reception of long data frames under high mobility conditions. In fact, although offline prototyping accelerates the design and testing of algorithms, it is important to understand its foundations and design particularities e.g., unconstrained computational and storage resources, unlimited precision using floating-point implementation, no need to account for real-life implementation constraints or the complexity of the control plane, perfectly synchronized signals or ideal channels are typically assumed. Thus it is clear that in order to achieve a thorough analysis of the implementation cost and feasibility of the target system (especially in scenarios requiring dynamic responsiveness or high mobility), real-time prototyping has to be employed.

- *Hardware/software partitioning*: the flexibility of non real-time prototyping in terms of resource requirements allows the designer to select an optimum hardware/software partitioning accounting for the implementation cost. It is a common practice to maintain the algorithms in MATLAB space, while only the RF section and the data capturing operates in real-time. Alternatively, a subset of the signal processing algorithms can be mapped to a DSP or a FPGA implementation, following a co-simulation or hardware-in-the-loop testing approach.
- *Hybrid prototyping*: the granularity of the prototyping strategy can be adjusted to fit the specific design and budget requirements. For instance a reduction in the prototyping complexity can be achieved by implementing/emulating more features in MATLAB-space or by making assumptions and system-wide simplifications. In hybrid prototypes a portion of the design resides in a computer simulation, while at the same time dedicated memory interfaces facilitate the communication with the bit-intensive portion of the design that runs on a FPGA device. This prototyping method downscales the real-time processing requirements in order to cope with the data-exchange constraints between the software and hardware processing domain.

1.2. The role of MATLAB in the design and validation process

As it has been described in the previous section, system-prototyping involving FPGAs and other specialized hardware equipment is subject to non idealities and certain signal impairments, which are not usually considered in a computer-based simulation (i.e., high-level models). Moreover, the heterogeneous hardware boards used for the prototyping of high performance real-time systems impose a series of hardware constraints in terms of processing capacity, available memory, maximum achievable clock frequency, I/O interfacing, DAC/ADC resolution and power consumption.

In the following sections, it will be shown how the previously described operating conditions and constraints can be either modelled or considered in MATLAB throughout the design and implementation process. The goal is to demonstrate the plural contribution of MATLAB in the FPGA-based rapid prototyping, beyond its well-established function as a high-level modelling tool:

I *Definition of system requirements*: Apart from its traditional operating perception, MATLAB can be used as a key companion throughout the analysis of system requirements in terms of computational resources and cost (e.g., implementation complexity, optimal hardware platform selection). Once the deployment scenario and specifications are

strictly defined (e.g., operating frequencies, channel bandwidth, channel specifications, sampling frequencies, DAC-ADC resolution) the high-level MATLAB model of the system can be modified to satisfy real-life system characteristics, according to the following key points:

- Account for system-wide signal impairments introduced by the complete hardware processing chain (baseband, RF and channel).
- Identify the most critical signal-processing blocks that play a definitive role in system's performance and computational load.
- Select the optimum algorithms satisfying a trade-off between resulting precision, hardware specifications and implementation complexity (e.g., required FPGA-re-sources).
- Adjust the data quantization at the different baseband processing stages.
- Account for the specifications, operation and functionality of the memory and control planes.

II *Co-simulation*: A very useful practice during the early stages of prototype development is to implement and simulate different parts of the target system using different design approaches and tools; i.e., one part of the system remains modelled and simulated in MATLAB, while the rest is designed using lower-level HDL simulation tools. The co-simulation of the differently modelled parts requires the communication of MATLAB with third party simulation environments. This can be realised by utilizing the data importing and exporting capabilities of MATLAB, or as it will be discussed later, by exploiting the interfaces of MATLAB with certain third-party tools. For instance, the prototyping of systems or algorithms using offline testbeds typically implies that the complex signal processing algorithms, and other emulated functionalities that serve the testing scenario, remain modelled in MATLAB. Using standard I/O functions, binary data can be read, written and quantized in MATLAB-space, providing a direct way to communicate with the remaining portion of the system which resides in an HDL-based simulation (i.e., using equivalent I/O connectivity options). The same co-simulation methodology can be used to test an algorithm, an independent processing block or a complete system designed in MATLAB against its HDL-based counterpart (designed in third party RTL simulation tools). This type of co-simulations have a critical contribution in the prototyping of real-life FPGA-based systems, because they provide the means to assess the fixed-point precision of the independent processing blocks comprising a digital baseband system and also because they produce reliable test vectors, which enable the performance validation of the RTL-algorithms.

III *Verification of the hardware-produced results*: MATLAB supports data importing and exporting in various formats and includes a series of pre-compiled libraries and mathematical functions. The latter facilitate the post-processing of data captured by baseband processing boards and assist the verification of the results produced by a FPGA-based prototype. The only requirement as far as the baseband signal processing platform is concerned is its ability to capture large amount of data in files that could be imported in MATLAB.

IV *Rapid-prototyping*: the previously described features and design-capacities of MATLAB are making it a prime candidate for the off-line prototyping and validation of wireless communication systems. Indeed, MATLAB plays a key role in off-line testbeds that are used to prototype state-of-the-art MIMO-OFDM systems [9–13].

The process of mapping a high-level MATLAB model to HDL logic and consequently to FPGA-based hardware is a complex and costly process, where many crucial decisions need to be taken. These include among others the environment where the system will be deployed, the expected operating conditions and the target implementation technology. MATLAB can be easily interfaced with third-party EDA tools and hardware equipment [14], a fact that facilitates this decision-making process. Additionally, the use of MATLAB in all prototyping-stages makes easier the interaction between different design-teams by providing a common working framework.

2. Design methodology

The design, implementation and on-board testing of high performance wireless communication systems under realistic conditions implies an undertaking with high stakes. Thus, the adoption of a well-structured design, implementation and validation methodology is a paramount requirement. The aim of this section is to offer an insight to a robust, yet generic, methodology, which demonstrates the contribution of MATLAB during the FPGA prototyping stages using a custom HDL design entry. The effectiveness of the proposed methodology is analysed using a practical case study, which involves the prototyping of a real-time MIMO mobile WiMAX system.

A fundamental guideline that applies throughout the design, implementation and on-board validation phases is a multi-stage testing strategy (Fig. 1). This starts from a baseband-to-baseband system testing under ideal conditions. The latter is performed both in simulation-time (MATLAB and consequently HDL-based) and at real-time in the target hardware platform using a direct connection of the transmitter and receiver. The scenario can then be augmented by adding the conversion stages (i.e., ADC and DAC). This implies re-simulating the MATLAB and HDL code and finally validate the FPGA implementation in real-time (i.e., connecting via a cable the output of the DAC device with the input of the ADC device). The final testing stage can be divided in two sub-stages; the first includes a direct cable connection of the RF front-ends and the second the inclusion of channel either by using antennas or a real-time channel emulator (both sub-stages can be priorly simulated in MATLAB and in HDL). This incremental testing approach allows the step-by-step characterization of the system.

2.1. Starting point

The development of a processing demanding real-time wireless communication system requires a wide range of skills, resources and time. A commonly accepted commencing point is the design of a baseline version of the target system, which complies with the following design requirements:

- *Modularity:* This feature facilitates the substitution, modification, extension and/or reuse of specific parts of the design.
- *Downscaled specifications:* The initial design-efforts should focus on the core signal processing algorithms and on the most critical aspects of the overall system architecture (e.g., high-throughput pipeline structures combined with efficient memory and control planes).

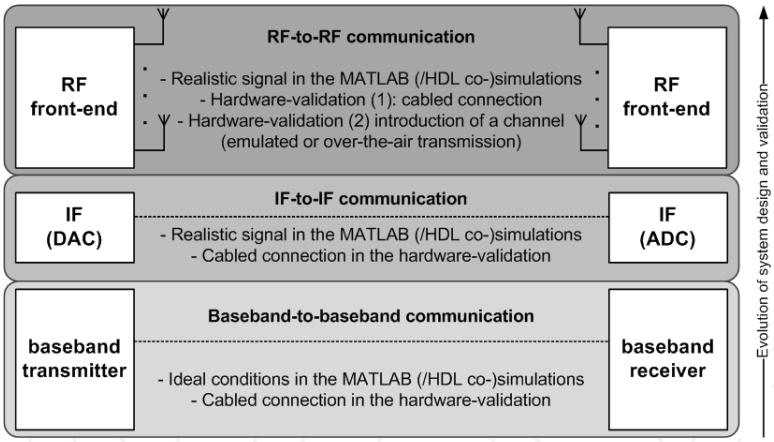

Figure 1. Multi-stage testing strategy

Once the baseline system-model is designed and validated, the proposed methodology can adjust the granularity of the system by accommodating more advanced features.

2.2. Proposed methodology

This section gives the details of the proposed design methodology, which is depicted in Fig. 2.

I *Basic transmitter modelling*: The first vital requirement for the design of any wireless communication system is the definition of the transmitted signal. The modelling of the transmitted signal is in most cases bound to the specifications of a wireless communication standard, which indicatively includes the OFDM parameters, the duplexing mode, the format and length of the frame, the number, value and location of the pilot tones, the guard-band size, the inter-carrier spacing, the available bandwidth sizes and the RF operating bands. At this initial stage the model of the transmitted signal is based on certain ideal conditions i.e., using floating-point logic, assuming unlimited processing resources during design-time and not accounting for signal-impairments (e.g., channel effects, noise).

II *Hardware-validation of the baseband transmitter model*: The output of the MATLAB model has two vectorial components, namely the in-phase and quadrature (I/Q). By writing the I and Q outputs in a MATLAB file (i.e., with file extension `.mat`), it is possible to make a direct hardware validation of the baseband transmitter model. As it was previously described, this is made feasible considering that numerous modern VSG instruments[1], provide the necessary API to download such files to an internal memory of the instruments. The latter with the help of an arbitrary waveform generator provides the real-time baseband digital I/Q signals, which then pass from the required DACs and RF conversion stages to produce the desired signal at the selected RF band. This is an indicative test and verification flow where MATLAB

[1] The described functionality is available, for instance, on the VSGs provided by Agilent (*http://www.agilent.com*) or Rode & Schwarz (*http://www.rohde-schwarz.com*). Further information on other hardware manufacturers supporting MATLAB communication may be found in *http://www.mathworks.com/products/instrument/hardware*.

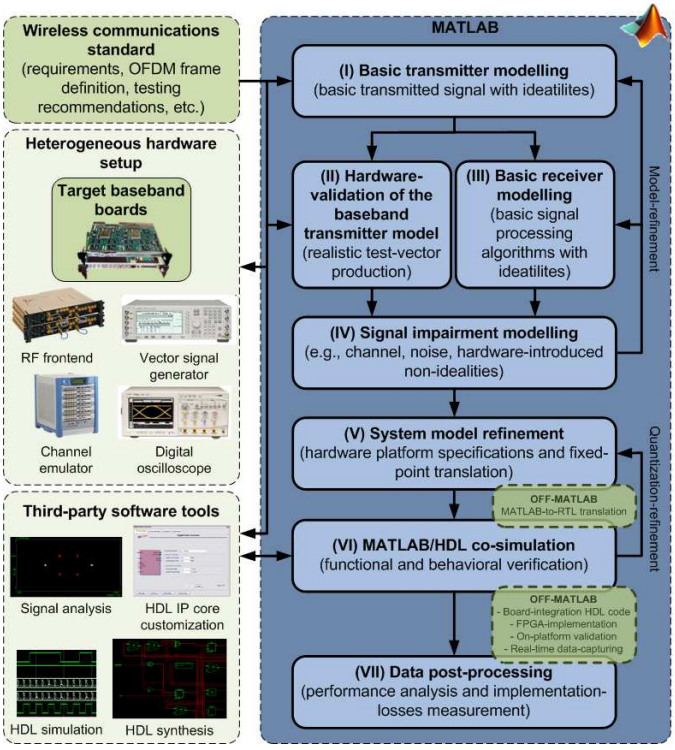

Figure 2. Proposed design, implementation and validation methodology

is directly interfaced with advanced instrumentation to produce a real-time signal. This signal can be then processed by specific testing hardware (e.g., signal analysers, digital oscilloscopes), which communicate with proprietary third-party software in order to perform standard-compliance tests. Additionally, the transmitted signal can be introduced via a cable connection to the receiver's RF down-converters (i.e., no channel should be used at this initial design stage) and after passing from the ADC stage at the receiver's acquisition boards, it can be retrieved using the FPGA devices and dedicated external memories of the baseband signal processing boards. The captured data constitute realistic test vectors that can be used for the development of the MATLAB model of the receiver, whereas the whole testing procedure permits a refinement of the initial transmitter model (see points IV and V). The testing setup described before could also include specialized equipment that add realistic signal impairments (e.g., real-time emulation of a selected channel, addition of noise or of Carrier Frequency Offset - CFO). However, such operating conditions make unreliable the capturing of test-vectors until the digital front-end of the receiver is developed and tested at the target FPGA board.

III *Basic receiver modelling*: The next step is the modelling of the signal processing algorithms at the receiver side. As in the case of the transmitter, the ideal MATLAB model of the receiver uses floating-point logic and does not have any design limitations in terms of processing and memory resources. The functional testing of the complete system is conducted by running a MATLAB simulation of the transmitter and receiver models

(i.e., ideal baseband-to-baseband signal). This could be extended by using the test vectors captured in the previous step (i.e., hardware-validated MATLAB model of the transmitter). Although the design is still not constrained by the limitations of the entire hardware processing platform, the performance of different algorithms could be studied, including those whose computational complexity makes their real-time prototyping challenging. The designer has therefore the opportunity to estimate the ideal performance of the overall system.

IV *Signal impairment modelling*: After finishing the ideal MATLAB model of the entire system, it is time to start adding real-world impairments. The latter are inherent features of hardware components and effects applied to analogue signals when propagating in physical mediums. This implies modifications of the originally designed MATLAB model to meet new operating conditions. The most indicative impairments that need to be modelled in MATLAB is the transmission over a defined channel, the addition of noise and CFO, the coupling of the baseband signal with the Local Oscillator (LO) and the introduction of a Direct Current (DC) level by the hardware platform. As a result, it is obtained a model of the transmitted signal that is significantly closer to real-world conditions. The signal processing algorithms at the receiver have to be modified and upgraded to account for these signal impairments.

V *System model refinement*: Additional modifications are required to the MATLAB baseband model of the system, before starting the challenging stage of mapping it to RTL code. The MATLAB models of the transmitter and receiver have to account for the the hardware platform specifications (i.e., ADC/DAC features, internal buses, I/Os, available FPGA-resources - including embedded memory and specialized digital signal processing - DSP - blocks, etc.). Thus, the signal processing algorithms must be refined as follows:

- *RTL-implementation awareness*: it is widely known that not all MATLAB structures or functions are implementable in an FPGA. Even if equivalent HDL constructs exist, they are used during simulation time but do not serve for logic synthesis (e.g., a for-loop construct with undefined number of iterations). Moreover, MATLAB includes several pre-compiled DSP functions (e.g., Fast Fourier Transform - FFT) and provides abstract arithmetic operators (i.e., the user calls the same operator independently of the type of the operands). For instance, the '$*$' operator provides the multiplication for integer, real or complex numbers, arrays and matrices. Although these MATLAB features provide a powerful workbench for users, it is common quite a mistake to underestimate the computational complexity and the internal arithmetic calculations of such operations, especially when they are meant to be mapped on a real-time RTL-based implementation (see example 2.1). It is therefore a key design requirement to evaluate the implementability and arithmetic complexity of the algorithms comprising the target system, in relation to the maximum processing and memory capacity of the target FPGA device. This usually gives a first idea of which design partitioning strategy can be followed (e.g., using various FPGA devices or a combination of FPGA devices and DSP microprocessors). The importance of this evaluation stage for the mapping of the MATLAB model to RTL code is crucial and may result in selecting different algorithms and lightweight versions of pre-compiled arithmetic functions. Another important task is to estimate the storage and intercommunication needs. This is made feasible by including in the MATLAB model a high-level representation of the memory and control planes.

- *Translation to fixed-point arithmetic*: the FPGA-based prototyping of wireless communication systems implies the use of fixed-point logic at baseband. This is a significant design constraint that has to be evaluated considering that MATLAB modelling is based by default on floating point arithmetic. In general terms the floating-point operations dramatically increase the FPGA logic utilization and result in lower clock speeds and longer pipelined structures when compared to fixed-point logic[2]. The designers are responsible for mapping the MATLAB algorithms to an HDL-based fixed-point logic, which in fact is a demanding and non-trivial task. The latter implies that all internal processing stages of the transmitter and receiver (both in MATLAB-space and HDL-design space) have to be appropriately simulated to tune them at an optimum fixed-point dynamic range, applying numerous truncation and scaling steps to achieve the best arithmetic precision. Additionally, each of the implemented HDL blocks has to be co-tested with the equivalent portion of the floating point Matlab model to ensure that the system performance is not compromised (see point VI). A very handy modification of the MATLAB model that assists the comparison with the equivalent RTL code is to apply quantization at the outputs of selected processing blocks that represent functional partitions of the design. This quantization process emulates the fixed-point logic.
- *Hardware constraints and specifications awareness*: the functionality of the MATLAB model of the transmitter and receiver can be further adapted to account for hardware-introduced constraints, brining it more close to real-life testing conditions. For instance, the MATLAB model can be adjusted to the Dynamic Range (DR) of the DAC/ADC circuitry of the target boards. The system DR depends on the modulation scheme, the modelled signal-impairments and the DAC/ADC specifications (i.e., number of bits of the produced samples and applicable amplifier gains). Additionally, a number of pre-compiled HDL IP cores used in the prototyping stage of FPGA-based DSP algorithms (e.g., FFT, Digital Down Converter - DDC, pipelined divider) are offering a limited range of input/output data-width options. This results in further quantization analysis, assuming that the reception of samples is scaled within a certain dynamic range. The on-board FPGA implementation entails a series of other design limitation, which are hard to be emulated at MATLAB space. Indicative examples of such hardware implementation features include the interfacing of the FPGA design with high-speed buses and the latencies introduced by several FPGA IP cores; the latter increase the intermediate storage requirements and add more complexity to the control plane.
- *Satisfy a trade-off between numerical complexity and system performance*: The system designer has to discover the optimal achieved performance of the designed system (i.e., baseband, RF and channel propagation stages) through a recursive process, taking into account the processing and memory resources of the target FPGA device, the additional inherent constraints of the hardware platform and the minimum required yield of the system. The latter is subject to specific prerequisites related to numeric precision, throughput and compliance with certain performance metrics (e.g., Bit Error Rate - BER, average data rate). This means that the MATLAB model will be adjusted until the designer achieves the desired performance, which eventually will allow him to pass to the next design stage of RTL coding.

[2] It is useful to mention that specific floating-point arithmetic libraries, Intellectual Property (IP) cores, embedded microprocessors and other dedicated processing components can be used in FPGA devices to serve the needs of particular applications that require this type of arithmetic operations [15, 16]

Example 2.1: let's consider the simple multiplication of two complex numbers, $a = 2.5 + 3.2i$ and $b = 1.7 - 4.5i$. In MATLAB a user would simply type '`c = a * b`' abstracting away the underlying calculation. However, when considering a RTL design many other aspects must be considered.
First, let us assume a dynamic range of input samples that satisfy the $(-8, 8)$ margin. Also let us consider a binary representation of samples with 16 bits, where 4 bits are used to represent the sign and the integer part and the remaining bits are used to represent the fractional part. The I and Q components of the complex numbers have to be represented separately. This can be modelled in MATLAB using the `quantizer`, `num2bin` and `bin2num` functions of the fixed-point toolbox:

```
q = quantizer([16 12]);
I_a = num2bin(q,real(a)); Q_a = num2bin(q,imag(a));
I_b = num2bin(q,real(b)); Q_b = num2bin(q,imag(b));
```

Furthermore, the complex multiplication has to be broken down to basic operations. In MATLAB this can be done as follows:

```
I_c = bin2num(q,I_a)*bin2num(q,I_b) - bin2num(q,Q_a)*
bin2num(q,Q_b);
Q_c = bin2num(q,Q_a)*bin2num(q,I_b) + bin2num(q,I_a)*
bin2num(q,Q_b);
```

A first approximation of the error introduced by quantization can be measured with a simple subtraction: `quant_loss_I = real(c) - I_c; quant_loss_Q = imag(c) - Q_c;`

Moreover, each arithmetic operation in RTL coding results in a bit-width grow: e.g., the multiplication of two *N*-bit operands results in *2N* bits and the addition of two *N*-bit operands results in *N+1* bits. Therefore, RTL coding implies that each of the previously described intermediate operations has to be considered separately:
`intermediate_op1 = I_a * I_b`, results in 32 bits
`intermediate_op2 = Q_a * Q_b`, results in 32 bits
`intermediate_op3 = intermediate_op1 - intermediate_op2`, results in 33 bits
To sum up, a quantization adjustment (i.e., bit-alignment in case the bit-width of the different operands grows differently) and/or a data-truncation will be required between the different intermediate calculations to limit the overall computational complexity. Additionally, in order to achieve a better timing performance of the FPGA design, the intermediate calculations of complex operations are placed in different clocked-processes: i.e., the calculation of '`intermediate_op3`' '`intermediate_op1`' and '`intermediate_op2`' would be placed in a different clocked process. Therefore, a latency of one clock cycle would be introduced at each intermediate calculation. Although the bit-width growth and the introduced latencies are not modelled in MATLAB, it is highly recommended to analyse such aspects in order to assess the system's complexity. A complex calculation may result in a change of the quantization, which in turn will require further modifications of the MATLAB model.

VI *MATLAB/HDL co-simulation*: Each portion of the implemented HDL code that forms a functional component of the system has to be co-simulated with the equivalent partition of the MATLAB model. This allows to assess both the functional correctness and the achieved performance. As already mentioned before, there are several ways to use the co-simulation methodology. An indicative example is when part of the system simulation resides in MATLAB space, whereas another portion is hosted in a third party HDL design tool; the output of the MATLAB model can be quantized and saved to a file, which can be inserted to the HDL-based simulation. The results produced by the HDL simulation can also be quantized and written to a file, which is fed back to MATLAB. This is a very useful way to verify the functionality and inter-working of the system that is implemented in different simulation domains. In addition, it also enables the evaluation of the precision achieved by the HDL model by comparing its performance with the non-quantized results produced by the MATLAB model. It is important to highlight the vital role of co-simulations for selecting an optimum quantization that satisfies a trade-off between precision and computational complexity. Finally, the MATLAB/HDL co-simulations provide the best means to evaluate and test HDL IP cores and common signal processing operations (e.g., optimizing the trade-off between implementation complexity and result precision requires the calculation and truncation of the produced outputs).

VII *Data post-processing*: Data can be captured at different baseband processing stages once the system (or parts of it) is implemented in a target FPGA board. This data can be inserted in MATLAB after using the proper quantization to enable the off-line calculation of the required performance metrics (e.g., BER, Signal-to-Noise Ratio - SNR, Error Vector Magnitude - EVM). MATLAB can also be used to automate the post-processing of the captured data-frames, and provide a reliable calculation of the desired performance metrics (i.e., mean value over thousands of data samples).

The end of a major design cycle is reached when the performance of the RTL prototype is finally validated on real-time hardware and does not require any further modifications. This gives the opportunity to the system designer to introduce additional features by iterating over the previously described methodology. The proposed incremental design approach implies a relative low effort to augment the features of a working prototype. This is mainly due to the fact that a modular and reusable code is already available, while at the same time the critical parts of the design and the system bottlenecks are well defined. The same applies to the hardware platform which is already thoroughly studied and characterized.

3. A practical case study

This final section presents a practical case study of the manifold contribution of MATLAB throughout the entire design, development and prototyping stages of a real-time mobile WiMAX system [3, 4]. The use-cases focus on the Single Input Single Output (SISO) configuration of the system [17] that features one antenna at the transmitter and receiver sides respectively. Taking as an exemplar basis the development of the SISO system, the presented incremental design methodology can be reused to develop the MIMO system, which however is not covered in this chapter. The main specifications of the target system are summarized in table 1.

The GEDOMIS® testbed (see Fig. 3), was used to prototype and validate the system described in this chapter. GEDOMIS® features multiple APIs, dedicated signal analysis software tools

Parameter	Value
Wireless telecommunication standard	IEEE 802.16e-2005
Antenna schemes: SISO, SIMO, MIMO	1x1, 1x2, 2x2
RF band (GHz)	2.595
IF (MHz)	156.8
Channel bandwidth (MHz)	20
Baseband sampling frequency (MHz)	22.4
ADC sampling frequency (MHz)	89.6
Cyclic prefix (samples)	512 (1/4 of the symbol)
Modulation type	QPSK
Duplex mode	TDD
FFT size	2048
OFDM symbols per frame	48
Supported permutation schemes	PUSC and AMC (DL)
Diversity scheme (2x2 MIMO)	Matrix-A (Alamouti)

Table 1. Basic OFDM and PHY-layer specifications of the described system.

and a heterogeneous hardware setup. The latter comprises signal generation equipment, multi-channel signal conversion boards, a real-time radio channel emulator and FPGA-based baseband signal processing boards [18]. The examples detailed in the remaining of the chapter do follow the previously proposed multi-stage testing strategy (see Fig. 1) and do not always require the use of the full set-up of this testbed.

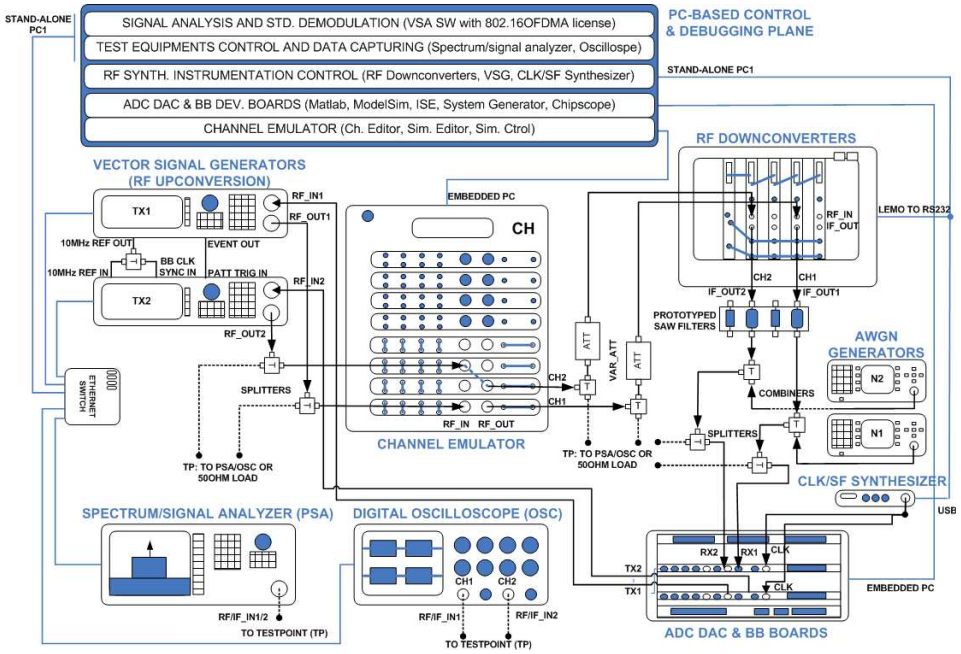

Figure 3. The GEDOMIS® testbed setup.

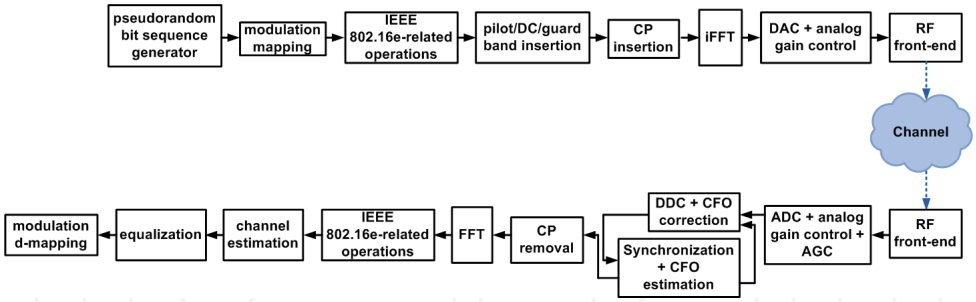

Figure 4. Basic architecture of the SISO transmitter and receiver systems.

3.1. PHY-layer prototype of a single antenna mobile WiMAX transceiver

Fig. 4 shows a simplified functional block diagram of the SISO mobile WiMAX system. Taking as a reference the design methodology presented before, this section gives an example of the MATLAB usage in each prototyping stage.

I *Basic transmitter modelling*: The first task is the accurate definition of the OFDM-based frame structure. Thus, it has to be identified the basic function of the different frequency subcarriers within each OFDM symbol. In our case, the frame is structured according to the Partial Used Subcarrier (PUSC) and the AMC subcarrier permutation schemes, which are defined in the mobile WiMAX standard [19]. The main characteristics of both OFDM symbol structures are summarized in table 2. Example 3.1 shows the MATLAB-modelling of the processing block responsible for inserting the pilot subcarriers, the DC and the guard bands, according to the mobile WiMAX specifications. The additional subcarrier organization and permutation operations required by the WiMAX standard can be easily designed in MATLAB-space. Finally, the use of a standard inverse FFT function provides the ideal I/Q baseband outputs of the transmitter (i.e., floating-point values).

Scheme	Parameter (per OFDM symbol)	Value
PUSC	Data subcarriers	1440
	Pilot subcarriers	240
	Null subcarriers	368
	Clusters	120
	Subcarriers per cluster	14
	Subchannels	60
	Data subcarriers per subchannel	24
AMC	Data subcarriers	1536
	Pilot subcarriers	192
	Null subcarriers	320
	Bands	48
	Bins per band	4
	Subcarriers per bin	9
	Subchannels	32
	Data subcarriers per subchannel	48

Table 2. Principal parameters of the PUSC and AMC permutation schemes.

Example 3.1: Fig. 5 shows the pilot distribution in the PUSC permutation scheme, described in the mobile WiMAX standard.

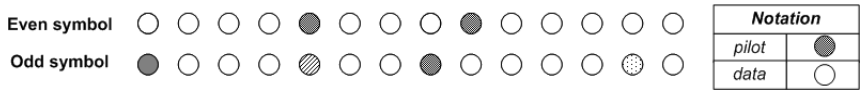

Figure 5. Location of the pilot symbols in the PUSC permutation scheme.

Taking into account this pilot distribution, the following MATLAB code represents the insertion of the pilot tones in a PUSC-structured OFDM frame:

```
%The PUSC-formatted OFDM symbols (i.e., outputs of the block in
charge of the IEEE 802.16e-related operations) are loaded in the
variable 'mWiMAX_PUSC_data'.
load('mWiMAX_PUSC_data')
PUSC_zone_length=30;

%There will be 240 pilots per OFDM symbol.
pilot=4/3+j*0; %Pilot value defined by the WiMAX standard.

%Each PUSC OFDM symbol contains 120 clusters of 12 contiguous
data subcarriers, where 2 pilots will be added.
evenSymb=1;
data_and_pilots=[];
for symb_index=0:PUSC_zone_length-1
symb_offset=symb_index*120*12;
ofdm_symbol=mWiMAX_PUSC_data(symb_offset+1:symb_offset+120*12);
cluster=[];
pilotCluster=[];

for cluster_index=0:119
subc_offset=cluster_index*12+1;
cluster=ofdm_symbol(subc_offset:subc_offset+11);
if evenSymb
pilotClus=[cluster(1:4) pilot cluster(5:7) pilot cluster(8:12)];
else
pilotClus=[pilot cluster(1:11) pilot cluster(12)];
end
ofdm_symbol=[ofdm_symbol pilotClus];
end

data_and_pilots=[data_and_pilots ofdm_symbol];
evenSymb=mod(evenSymb+1,2);
end

%A total of 368 null subcarriers are inserted.
ofdm_symbol=[zeros(1,184) ofdm_symbol(1:840) 0
ofdm_symbol(841:1680) zeros(1,183)];
```

II *Hardware-validation of the baseband transmitter model*: As already described before, the I/Q output vectors of the MATLAB model of the ideal transmitter can be stored (separately) in a MATLAB file. The latter can be downloaded to an internal memory of a VSG instrument (as described in Example 3.2). The VSG can be programmed to use these vectors in order to produce a real-time RF signal. This is made feasible by exploiting its embedded arbitrary waveform generator, DAC devices and RF upconversion circuitry. The validation of this signal using third party software tools and hardware instruments is very important, considering that several signal impairments and hardware constraints can be identified during early design stages.

Example 3.2: The first step for the prototyping and testing of the ideal transmitter MATLAB model using off-line testbed principles, requires the storage of the output I/Q components in two files.

```
%The frequency-domain data produced by the baseband transmitter
(i.e., before the IFFT) is loaded in the variable 'BB_data'.
load('BB_data')

%A short silence period precedes each frame.
silence_length=2560*5;
transmitted_signal=zeros(silence_period_length,1);

%The frame is composed by a preamble and 48 OFDM symbols.
for symb_index = 1:49
BB_ofdm_symbol=BB_data(2048*(symb_index-1)+1:(2048*symb_index));
%Conversion from frequency to time domain.
time_ofdm_symbol=zeros(512+2048,1);
time_ofdm_symbol(513:end)=ifft(BB_ofdm_symbol);

%Inclusion of the CP (i.e, the CP is a copy of the last 512
symbols of the OFDMA symbol).
preamble_length=512;
time_ofdm_symbol(1:preamble_length)=time_ofdm_symbol(1537:end);
transmitted_signal=[transmitted_signal; time_ofdm_symbol];
end

%I/Q component extraction.
custom_Tx_I=real(transmitted_signal);
custom_Tx_Q=imag(transmitted_signal);

%Creation of the 'custom_Tx_frame.mat' file to stimulate the VSG.
save('custom_Tx_frame.mat', 'custom_Tx_I', 'custom_Tx_Q');
```

Fig. 6, 7 and 8 show the configuration of the Agilent Signal Studio Toolkit. The latter is the software programming interface used to configure an Agilent E4438C VSG with the `'custom_Tx_frame.mat'` file.

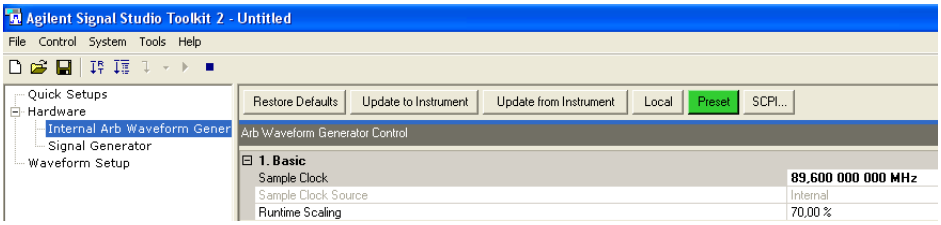

Figure 6. Agilent Signal Studio Toolkit configuration: ADC sampling frequency.

The principal parameters that need to be defined by the user to properly conduct the hardware-validation of the ideal transmitter, are the DAC sampling frequency, the desired RF band and the names of the variables of the MATLAB-generated file containing the I/Q components. The VSG is then able to apply the required IF-to-RF upconversion and provide a realistic RF signal.

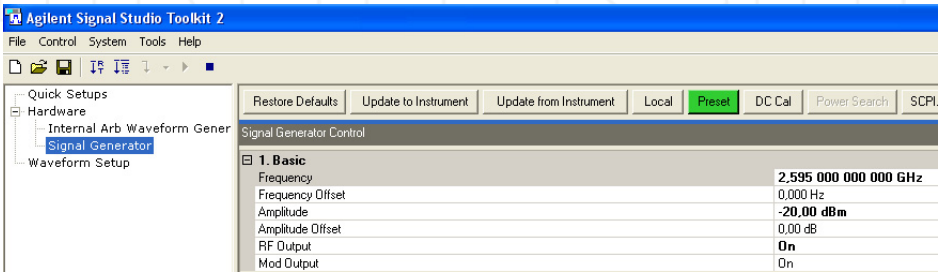

Figure 7. Agilent Signal Studio Toolkit configuration: RF band.

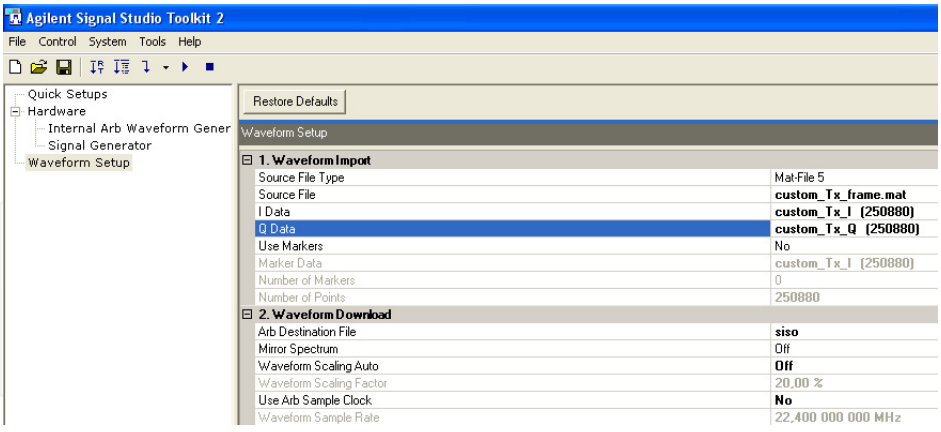

Figure 8. Agilent Signal Studio Toolkit configuration: loading of the I/Q components of the MATLAB model of the ideal transmitter.

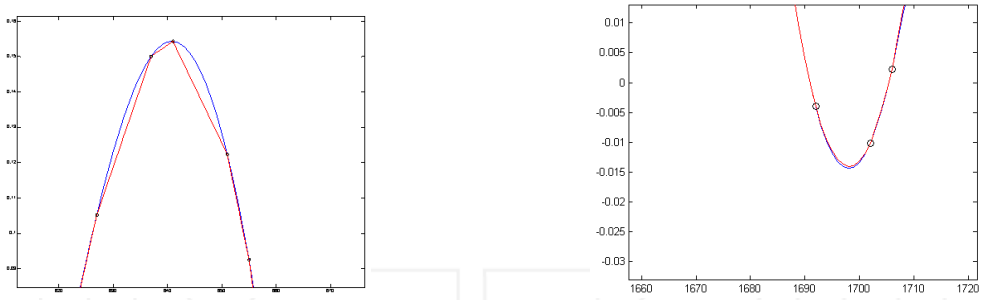

Figure 9. As it can be observed, the deviation between the ideal and the estimated channel is notable when using a linear interpolation, which accounts only for the two closest pilots for each subcarrier. On the contrary, the obtained results are better when applying a quadratic interpolation approach which uses three neighbouring pilots in the calculations.

Using a RF-to-RF cable connection data can be captured at the receiver baseband boards. This provides realistic test vectors that will be later used to design and debug the MATLAB model of the receiver.

III *Basic receiver modelling*: The first step is to model the ideal received signal. In this sense, it is assumed that the received signal is identical with the transmitted one, without making use of quantizations or accounting for signal-impairments. In other words, an ideal baseband-to-baseband communication is modelled as follows:

$$c(t) = \tilde{x}(t), \tag{1}$$

where $x(t)$ represents the equivalent transmitted baseband signal.

The modelling of the receiver is based on WiMAX-defined processing functions (e.g., permutation of the subcarriers) and common signal processing operations (e.g., FFT). MATLAB provides the ideal modelling environment to compare the performance tradeoffs of different signal processing algorithms. As an example, Fig. 9 shows the comparison of a linear and a quadratic interpolation for a pilot-based channel estimation algorithm. This type of algorithm design and benchmarking allows the designer to make early decisions tailored for the specifications of the target hardware platform. Nonetheless, the validation of the critical parts of the receiver, such as the synchronization or the channel estimation requires a signal model that is closer to real-world conditions (i.e., accounting for impairments and hardware constraints). Once this modified version of the received signal is available, the designer is able to make a precise selection of algorithms that are suitable for the anticipated channel conditions and the characteristics of the target hardware platform.

IV *Signal impairment modelling*: Having already modelled the ideal system, the next step is to modify the signal model to include the expected signal impairments. This requires the analysis of the main specifications and performance of the target hardware. In the test-case described herein, certain signal impairments such as the I/Q gain and phase imbalances or LO drifts are ignored because of the performance indicators and specifications of the equipment comprising the GEDOMIS® testbed. It is important however that each designer exhaustively examines the complete set of potential signal impairments and ignore only those that have negligible effects to the received signal. This procedure is subject to generic signal processing and propagation principles, but

also requires a hardware-specific analysis of potential impairments (i.e., different for each testbed). In our case, the resulting refined received signal model at the output of the RF down-converters can be expressed as follows:

$$c(t) = \Re\{x(t) \cdot e^{j2\Pi(f_{IF}+\Delta f)t}\} + A \cdot cos(2\Pi(f_{IF} + \Delta f)t + \varphi) + w(t), \quad (2)$$

where $x(t)$ represents the useful part of the received baseband signal, f_{IF} is the Intermediate Frequency (IF), Δf is the Carrier Frequency Offset (CFO), $A \cdot cos(2\Pi(f_{IF} + \Delta f)t + \varphi)$ represents the unwanted residual carrier located at the center of the useful signal-spectrum (i.e., introduced by the LO coupling at the transmitter) and, finally, $w(t)$ is the Gaussian noise. The useful part of the received baseband signal can be expressed as:

$$x(t) = \tilde{x}(t) \star H(t), \quad (3)$$

where $\tilde{x}(t)$ is the equivalent transmitted baseband signal and $H(t)$ is the equivalent baseband of the time impulse response of the channel between the transmit and receive antennas. Example 3.3 shows the MATLAB model of the refined signal shown in equation (2). Additionally, other aspects related to the RF transmission, reception and downconversion of the signal are also contemplated (e.g., oversampling).

Example 3.3: MATLAB code for the signal impairment modelling.

```
%A custom IFFT function, providing an oversampled output is
required (i.e., the ADCs are oversampling by 4).
function samples = ifft_x4oversamp(BB_ofdm_symbol)
temp_symbols = zeros(8192,1);
temp_symbols(1:1024) = BB_ofdm_symbol(1:1024);
temp_symbols((8192-1024+1):8192) = BB_ofdm_symbol(1025:2048);
samples = ifft(temp_symbols,8192);
%----- function end -----

%To simulate the channel a coefficients file will be used.  The
channel will be loaded in the variable 'channel'.
load('channel_coefficients')
%The frequency-domain data produced by the baseband transmitter
(i.e., before the IFFT) is loaded in the variable 'BB_data'.
load('BB_data')

%A short silence period precedes each frame:  now we have to
account for the over-sampling of the ADCs.
silence_length=2560*5*4;
transmitted_signal=zeros(silence_period_length,1);

%The frame is composed by a preamble and 48 OFDM symbols.
for symb_index = 1:49
BB_ofdm_symbol=BB_data(2048*(symb_index-1)+1:(2048*symb_index));
%Introduction of the LO coupling (i.e., DC carrier is not 0).
BB_ofdm_symbol(1)=2;
```

```
%Conversion from frequency to time domain.
time_ofdm_symbol=zeros(2048+8192,1);
time_ofdm_symbol(2049:end)=ifft_x4oversamp(BB_ofdm_symbol);
%Inclusion of the CP (i.e, the CP is a copy of the last 512
symbols of the OFDMA symbol - oversampled by 4).
preamble_length=512*4;
time_ofdm_symbol(1:preamble_length)=time_ofdm_symbol(8193:end);
transmitted_signal=[transmitted_signal; time_ofdm_symbol];
end

%The modelled CFO will be equivalent to a third of the
intercarrier separation.
eps_freq=-1/3;

%Convolution of the frequency domain signal with the channel and
inclusion of the CFO and the noise.
SNR=25;
first_sample=silence_length+preamble_length+1;
mean_power = mean(abs(transmitted_signal(first_sample:end)).^2);
noise_power = mean_power/(10^(SNR/10));
received_signal = conv(transmitted_signal,channel);
rand_I=randn(size(received_signal));
rand_Q=randn(size(received_signal));
received_signal = received_signal+sqrt(noise_power/2)*(rand_I +
j*rand_Q);
received_signal = real(received_signal.*exp(j*2*pi*((156.8 +
(eps_freq*22.4/2048))*(1:length(received_signal))'/89.6)));
```

V *System model refinement*: In order to have a MATLAB model that provides a close match to the prerequisites of RTL coding, further modifications and refinements have to be conducted. This principally involves the emulation of fixed-point arithmetic in specific outputs of the MATLAB model. The trade-off between resulting precision and computational complexity has to be investigated. The more bits used to represent the fixed-point data, the more precision is achieved in the arithmetic operations. Considering that the prototyping target is a high performance real-time wireless communication system, it is required to use additional bits for the representation of signals, which consequently increases the FPGA processing and memory requirements. Different quantizations can be tested to analyse their effect both on independent processing stages, as well as on the overall system performance. An indicative example is when 16-bit words are used for the intermediate calculations of a custom MATLAB FFT function, featuring a radix-2 butterfly structure. This results in an aggregate quantization loss of $87 \cdot 10^{-2}$. The equivalent loss when using 32-bit words is reduced down to $13 \cdot 10^{-7}$. By inserting the quantized results to each of the remaining processing stages of the signal processing chain, it can be evaluated the performance-loss of the overall system. Hence, retaking the example mentioned before, the 16-bit quantized outputs of the FFT result in a performance degradation of the channel estimation (i.e., the precision-loss of

the estimated coefficients for the pilot tones increases the error during the interpolation stage). Example 3.4 presents the modified version of the MATLAB signal, which accounts for hardware constraints and applies the desired quantization. It is assumed a 14-bit ADC, a QPSK modulation and a value of the pilot signals of $\pm 4/3$ (defined in the WiMAX standard). For this testing scenario the DR is set to $[-1.9, 1.9]$: i.e., 2 bits represent the sign and the integer part and the remaining bits represent the fractional part.

VI *MATLAB/HDL co-simulation*: this section gives representative examples of the MATLAB versus HDL co-simulations, which is a vital procedure that has to be applied in all FPGA prototyping cases. Continuing from the previous example, the digitized IF signal at the receiver (i.e., ADC outputs) will be processed by the DDC component, which comprises a programmable digital synthesizer and a complex Finite Impulse Response (FIR) lowpass filter that eliminates out-of-band components. The input signal at the DDC is multiplied with a sine and a cosine (produced by the digital synthesizer). This multiplication results in the I and Q vector components, which are finally filtered and decimated in order to produced the desired baseband signal. This procedure is considered a key functionality of the Software Defined Radio (SDR). The digital synthesizer can be tuned on-the-fly by accessing a digitally-controlled register. This fact allows designers to correct the CFO, an inherent impairment of real-life RF front-end systems.

Example 3.4: MATLAB code that models the constraints introduced by the utilization of a particular ADC.

```
ADC_quantization=quantizer([14 12]);
gain=1.9/max(abs(received_signal));
ADC_samples = (received_signal.*gain)';
ADC_samples_binary = num2bin(ADC_quantization, ADC_samples);
ADC_samples_quantized = bin2num(ADC_quantization,
ADC_samples_binary);
```

Example 3.5 describes how the Xilinx DDC IP core was configured using MATLAB. In more details, we have used the Filter Design and Analysis Tool (`fdatool`), to design the required low-pass filter and produce the filter coefficients required for configuring the DDC core. Considering the importance of the DDC for the correct operation of the receiver, the MATLAB versus HDL co-simulation provided a crucial contribution for the evaluation of the fixed-point precision and guided the tuning of the configurable parameters featured in the DDC IP core.

Example 3.5: the SISO mobile WiMAX receiver requires the design of a low-pass filter with a decimation stage (denoted as polyphase decimator filter in the DDC IP). The configuration parameters of the `fdatool` are shown in Fig. 10.

The resulting filter has 103 coefficients, which can be quantized and exported to a file (i.e., with file extension `.coe`). The latter can be used to configure the Xilinx DDC IP core, as depicted in Fig. 11.

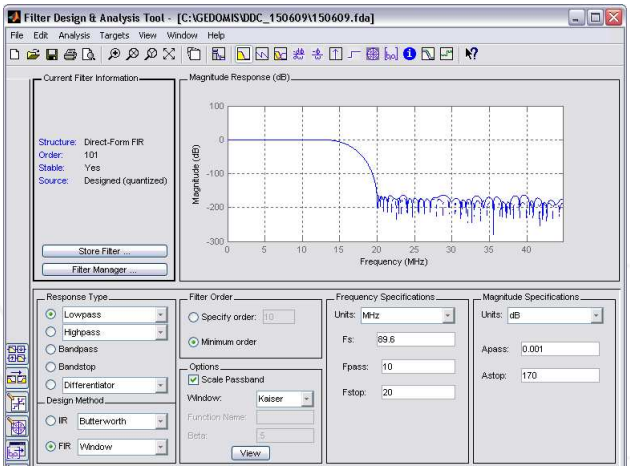

Figure 10. Utilization of the `fdatool` to design a FIR low-pass filter.

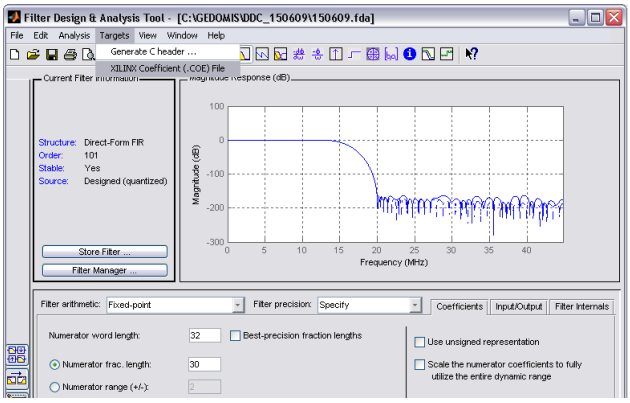

Figure 11. Exporting the FIR low-pass filter coefficients.

The code of the equivalent MATLAB model of the DDC, using the coefficients generated by `fdatool` is the following:

```
%The 'eps_freq' parameter represents the estimated CFO, as
returned by the synchronization block.
function baseband_signal = DDC(ADC_samples, eps_freq)

%The FIR-coefficients generated with 'fdatool' are loaded onto
the 'hfilter' variable.
load('hfilter')

%Modelling of the DDC functions (including CFO-correction).
cos_samples = ADC_samples.*cos(2*pi*(22.4-eps_freq*22.4/2048)*
(1:length(ADC_samples))/89.6);
```

```
sin_samples = -ADC_samples.*sin(2*pi*(22.4-eps_freq*22.4/2048)*
(1:length(ADC_samples))/89.6);
filter_cos = conv(cos_samples,hfilter);
filter_sin = conv(sin_samples,hfilter);
baseband_signal = filter_cos(1:4:end)+j*filter_sin(1:4:end);
%----- function end -----
```

Fig. 12 shows how the digital filtering stage of the Xilinx DDC IP can be configured using the coefficients file produced in MATLAB.

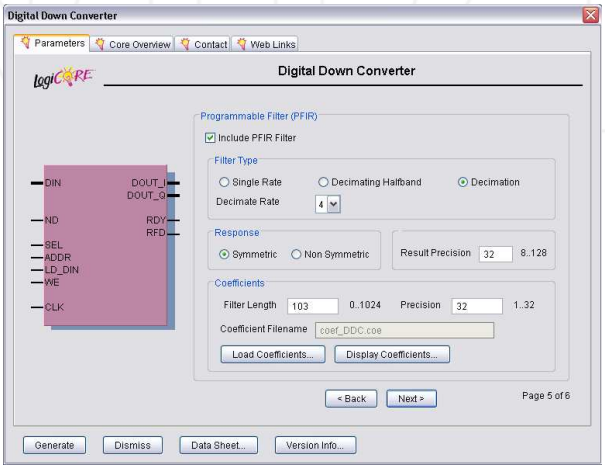

Figure 12. Configuration of the digital filtering stage of the DDC IP core using the MATLAB-generated coefficients.

Example 3.6 covers the main steps required to verify the behaviour and performance of an independent processing block (built using HDL code), by interfacing it with the MATLAB model of the remaining components of the system. The one-to-one comparison of the HDL model with its MATLAB counterpart provides a reliable analysis of the implementation losses (i.e., fixed-point versus floating point) and facilitates the selection of an optimum quantization (i.e., trade-off between precision and computational complexity, optimization of the bit-alignment and truncation operations).

Example 3.6: The MATLAB and VHDL code of the DDC processing stage that is required to run the co-simulation is quoted next. The output of the MATLAB model is written to a file, which is later used as a test vector of the RTL-simulation:

```
%The quantized outputs of the ADC are written to a file, which
will be used to stimulate the DDC IP core.
fileOut=fopen('DDC_core.in','w');
```

```
for k = 1:length(ADC_samples_quantized)
DIN = num2bin(ADC_quantization,(ADC_samples_quantized(k)));
fprintf(fileOut,'%s\n',DIN);
end
fclose(fileOut);
```

In the following simplified version of the VHDL code of the DDC block, the MATLAB-generated signal will be used as input to the RTL code. Additionally, the results produced by the HDL simulation are written to a file. This is used in recursive manner for the MATLAB simulation of the remaining processing blocks of the receiver:

```
-The quantized outputs of the ADC are read from a file and used
as inputs to the DDC IP core.
FILE inputFile :  TEXT OPEN READ_MODE IS "DDC_core.in";

-The RTL-generated outputs are written to a file, which will be
used to stimulate the MATLAB model.
FILE outputFile :  TEXT OPEN WRITE_MODE IS "DDC_RTL.out";

-Instantiation of the Xilinx DDC IP core.
DDC_core_ins :  DDC_core PORT MAP (
 -Input ports
CLK => clock_adc,
SEL => reset,
DIN => data_in,
ND => data_valid_in,
LD_DIN => prog_DDS_value,
ADDR => prog_DDS_addr,
WE => prog_DDS_write_enable,
 -Output ports
RFD => ready_for_data,
RDY => output_ready,
DOUT_I => BB_I_comp,
DOUT_Q => BB_Q_comp);

-Read MATLAB-generated data from a file to stimulate the DDC.
PROCESS
VARIABLE L_IN : LINE;
VARIABLE DATA : STD_LOGIC_VECTOR(13 DOWNTO 0);
BEGIN
reset <= '1';
data_in <= (others => '0');
data_valid_in <= '0';
prog_DDS_addr <= cnt_DDCreg;
WAIT FOR 44.64 ns;
reset <= '0';
data_valid_in <= '1';
```

```
FOR k IN 0 TO cnt_lengthDDC LOOP
READLINE(inputFile,L_IN);
READ(L_IN, DATA); data_in <= <= DATA;
WAIT FOR 11.16 ns;
END LOOP;
WAIT;
END PROCESS;

- Write the RTL-results to a file to stimulate the MATLAB model.
PROCESS(clock_adc)
VARIABLE L_OUT : LINE;
BEGIN

IF RISING_EDGE(clock_adc) THEN
IF output_ready = '1' THEN
WRITE(L_OUT, BB_I_comp);
WRITELINE(outputFile, L_OUT);
WRITE(L_OUT, BB_Q_comp);
WRITELINE(outputFile, L_OUT);
END IF;
END IF;
END PROCESS;
```

When comparing the 32-bit words at the output of the HDL-based DDC processing block with the equivalent stage of the MATLAB model, we realize that we may truncate this word to 16 bits with negligible precision losses. The required quantization is also obtained during this stage. Finally, in order to use the HDL-generated outputs in the MATLAB-simulation of the remaining blocks of the receiver, the following MATLAB code is required:

```
-The RTL-outputs of the DDC are read from a file and used as
inputs to the MATLAB receiver.
fileIn=fopen('DDC_RTL.out','r');
VHDLResult=fscanf(fileIn,'fclose(fileIn);

DDC_quantization=quantizer([32 26]);
BB_I_comp=[];
BB_Q_comp=[];
k=1;
for l=1:(cnt_lengthDDC+1)/2
binTmp=VHDLResult(k:k+31);
BB_I_comp(l)=bin2num(DDC_quantization,binTmp);
k=k+32;
binTmp=VHDLResult(k:k+31);
BB_Q_comp(l)=bin2num(DDC_quantization,binTmp);
k=k+32;
end
```

```
-The simulations show that a truncation to 16 bits can be applied
(i.e., q([16 14]) -> (27 DOWNTO 12) in RTL).
BB_I_trunc=[];
BB_Q_trunc=[];
for l=1:(cnt_lengthDDC+1)/2
BB_I_trunc(l)=BB_I_comp(5:20);
BB_Q_trunc(l)=BB_Q_comp(5:20);
end
```

VII *Data post-processing*: Once the FPGA-based prototype presents a stable operation, data can be captured in different parts of the system to evaluate its performance. This data could then be processed in MATLAB to calculate the desired metrics. When the goal is to characterize the performance of a system under mobility conditions, hundreds of data captures (e.g., generated with different channel seeds) containing several complete frames have to be captured and processed under different operating conditions (e.g., modify the SNR). MATLAB can be used to automate the calculation of the performance metrics, as shown in example 3.7.

Example 3.7: A simplified version of a MATLAB function, which automates the calculation of the EVM. The function relies on a predefined name-structure for reading the files:

```
%Function to automate the calculation of the EVM from real-time
post-equalization data captures.
function automatic_EVM_calculation(channel_spec, initial_rep,
final_rep, SNR_steps)

EVM_experiment=[];

for repetition = initial_rep:final_rep
for scenario = 1:SNR_steps
%Generation of a predefined 'file_name'
file_name=['postequal_' channel_spec '_' SNR_step '_'
num2str(repetition)];
%Call to the function calculating the EVM
EVM_array=EVM_calculation_HW_capture(file_name);
%Calculation of the mean value, conversion to dB and storage
EVM_experiment(index,scenario)=10*log10(mean(EVM_array));
end
index=index+1;
end

save(['postequal_' channel_spec '_' SNR_step '_'
num2str(initial_rep) '_to_' num2str(final_rep)
'.mat'],'EVM_experiment');
%----- function end -----
```

The EVM calculation of the FPGA-based prototype is made possible by comparing the files captured in the equalization block with the equivalent ones of the ideal MATLAB receiver. To achieve this we have to feed the MATLAB model with all the different test vectors captured in the post AGC stage of the FPGA prototype and produce the same amount of files at the output of the equalization stage. For each OFDM symbol in each captured frame, we would apply the following MATLAB operations (a mean value for each captured-file has to be calculated in the end):

```
deviation=[];
for index=1:cnt_data_carriers_per_symbol
deviation(index)=equal_out_RTL(index)-ideal_equal(index);
end
EVM_symbol=mean(abs(deviation).^2)/1;
```

4. Conclusion

The message that this chapter intended to convey is that MATLAB is having nowadays a diverse usage that goes beyond its initial conception as a generic mathematic modeling environment. Its functionality is valuable because it can be directly interfaced with various third party software/hardware design tools and instruments. Moreover, MATLAB has a multi-level contribution in the conceptual high-level modeling of a system, and it is an ideal candidate for rapid prototyping, since it can emulate the baseband signal processing when used in instrumentation-based offline testbeds. MATLAB is also used to emulate real-life hardware constraints and it can be adapted to serve HDL co-simulations. Its role is particularly important for the prototyping of bit-intensive systems such as the PHY-layer of modern wireless communication systems. This chapter proposed a comprehensive design methodology and quoted indicative examples, in order to highlight the previously mentioned benefits of MATLAB. In concrete, this chapter provided a guideline for the use of MATLAB during the prototyping of a FPGA-based real-time transceiver based on the mobile WiMAX standard. Finally, its critical contribution was contemplated by quoting extracts of the source code of the previously mentioned system prototyping phases.

Acknowledgements

The research leading to the published work was partially supported by the European Commission under projects BuNGee (248267) and BeFEMTO (248523); by the Catalan Government under grant 2009 SGR 891; and by the Spanish Ministry of Economy and Competitiveness under projects TEC2011-29006-C03-01 (GRE3N-PHY) and TEC2011-29006-C03-02 (GRE3N-LINK-MAC).

Author details

Oriol Font-Bach, Nikolaos Bartzoudis and David López Bueno
Centre Tecnològic de Telecomunicacions de Catalunya (CTTC), Spain

Antonio Pascual-Iserte
Centre Tecnològic de Telecomunicacions de Catalunya (CTTC), Spain
Department of Signal Theory and Communications, Universitat Politècnica de Catalunya (UPC), Spain

5. References

[1] *The MathWorks, Simulink*. http://www.mathworks.com, 2012.
[2] *Xilinx, System Generator for DSP*. http://www.xilinx.com, 2012.
[3] O. Font-Bach, N. Bartzoudis, A. Pascual-Iserte, and D. López Bueno. A Real-Time MIMO-OFDM Mobile WiMAX Receiver: Architecture, Design and FPGA Implementation. *Elsevier Journal of Computer Networks*, 55(16):3634–3647, November 2011.
[4] O. Font-Bach, N. Bartzoudis, A. Pascual-Iserte, and D. López Bueno. A Real-Time FPGA-based Implementation of a High-Performance MIMO-OFDM Mobile WiMAX Transmitter. In *Proc. International ICST Conference on Mobile Lightweight Wireless Systems (MobiLight)*, May 2011.
[5] M. Rupp, S. Caban, and C. Mehlführer. Challenges in Building MIMO Testbeds. In *Proc. European Signal Processing Conference (EUSIPCO)*, September 2007.
[6] A. Engel, B. Liebig, and A. Koch. Feasibility Analysis of Reconfigurable Computing in Low-Power Wireless Sensor Applications. In *Proc. International Symposium on Applied Reconfigurable Computing (ARC)*, March 2011.
[7] A. Sghaier, S. Areibi, and R. Dony. Implementation Approaches Trade-Offs for WiMax OFDM Functions on Reconfigurable Platforms. *ACM Transactions on Reconfigurable Technology and Systems*, 3(3):12:1–12:28, September 2010.
[8] M. Fernandez and P. Abusaidi. Virtex-6 FPGA Routing Optimization Design Techniques. White paper, Xilinx, October 2010.
[9] S. Caban, C. Mehlführer, R. Langwieser, A. L. Scholtz, and M. Rupp. Vienna MIMO Testbed. *EURASIP Journal on Applied Signal Processing*, 2006, 2006.
[10] S. Hu, G. Wu, Y. L. Guan, C. L. Law, Y. Yan, and S. Li. Development and Performance Evaluation of Mobile WiMAX Testbed. In *Proc. IEEE Mobile WiMAX Symposium*, March 2007.
[11] D. Ramírez, I. Santamaría, J. Pérez, J. Vía, J. A. García-Naya, T. M. Fernández-Caramés, H. J. Pérez-Iglesias, M. González-López, L. Castedo, and J. M. Torres-Royo. A comparative study of STBC transmissions at 2.4 GHz over indoor channels using a 2 x 2 MIMO testbed. *Wireless Communications and Mobile Computing, John Wiley and Sons*, 8(9):1149–1164, November 2008.
[12] G. Wang, B. Yin, K. Amiri, Y. Sun, M. Wu, and Jo. R. Cavallaro. FPGA Prototyping of a High Data Rate LTE Uplink Baseband Receiver. In *Proc. Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, November 2009.
[13] M. S. Khairy, M. M. Abdallah, and S. E. D. Habib. Efficient FPGA Implementation of MIMO Decoder for Mobile WiMAX System. In *Proc. IEEE International Conference on Communications (ICC)*, June 2009.
[14] *The MathWorks, Instrument Control Toolbox*. http://www.mathworks.com, 2012.
[15] M. Parker. Taking Advantage of Advances in FPGA Floating-Point IP Cores. White paper, Altera, October 2009.
[16] T. Vanevenhoven. High-Level Implementation of Bit- and Cycle-Accurate Floating-Point DSP Algorithms with Xilinx FPGAs. White paper, Xilinx, October 2011.

[17] O. Font-Bach, N. Bartzoudis, A. Pascual-Iserte, and D. López Bueno. Prototying Processing-Demanding Physical Layer Systems Featuring Single Or Multi-Antenna Schemes. In *Proc. European Signal Processing Conference (EUSIPCO)*, September 2011.

[18] *CTTC, GEDOMIS® testbed*. http://engineering.cttc.es/gedomis, 2012.

[19] IEEE 802.16e-2005. IEEE Standard for Local and Metropolitan Area Networks. Part 16: Air Interface for Fixed Broadband Wireless Access Systems. Amendment 2: Physical and Medium Access Control Layer for Combined Fixed and Mobile Operation in Licensed Bands, 2005.

INTECH