United Business Media EE Times europe Global news for the creators of technology

Modelling a high data-rate 802.16 wireless modem

Thierry Saunier & Vincent Perrier
(02/12/2009 8:04 AM EST)
URL: http://eetimes.eu/wireless/213000214

THALES has used CoFluent Studio to model an 802.16 modem using the orthogonal frequency division multiplexing (OFDM) modulation techniques to identify adaptations that improve the robustness of the links and/or the output data rate.

As part of the ITEA MARTES European research project, THALES needed to model a high data-rate 802.16 wireless modem to predict system behavior and performance though system-level modeling and simulation of partial hardware and software, perform design space exploration early in the lifecycle with embedded system executable specifications, and capitalize on the organization's co-design know-how at a higher abstract level to reduce development cost.

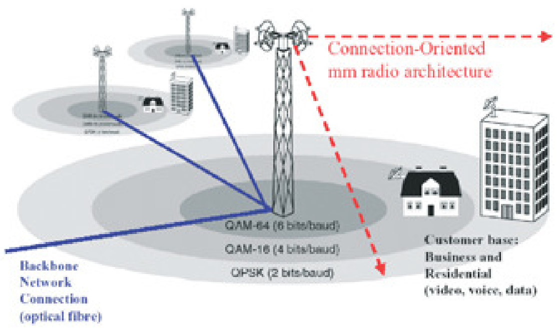


Fig 1: Network connections.

THALES used CoFluent Studio to model an 802.16 modem using the orthogonal frequency division multiplexing (OFDM) modulation techniques to identify adaptations that improve the robustness of the links and/or the output data rate. The modem contains an existing piece of software originally written in C++. It is modular code and uses three freeware libraries and an internal matrix library.

The wireless local area network (WLAN) is a network standard that allows the creation of local wireless networks using free radio frequencies in the 2.4 GHz and 5 GHz spectrum. With a range of few hundred meters, most WLANs are based on the popular 802.11 standard.

Wireless metropolitan area networks (WMAN) of 10 km range are based on the 802.16 standard which addresses frequencies below 11 GHz (non-directional environment, several antennas on large periphery) and on the 802.16a standard for frequencies up to 10 GHz (directional environment). WMAN could potentially compete with 802.11-based solutions in low mobility.

For the MARTES project, THALES modeled a system composed of a transmitter and a receiver as shown in the figure 2.

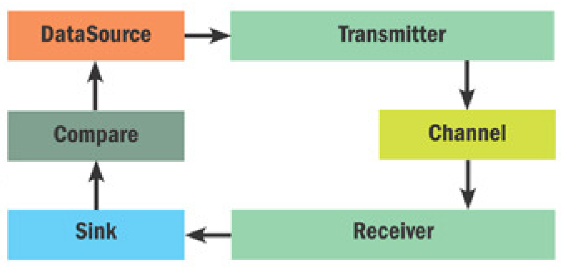


Fig 2: A system composed of a transmitter and a receiver.

The DataSource function generates series of bursts of unpredictable bit matrixes. Each bit matrix is then transformed by the Transmitter function, which simulates the processing of a radio device. The Transmitter outputs a matrix of real, complex values, which represent the signal emitted by the antenna. The real part of the values constitutes the I (inphase) channel, and the complex part the Q (quadrature) channel.

These two channels are applied to an analog modulator. The behavior of the analog modulator is not taken into account in the model. The channel function performs operations to simulate the effects of the air channel on the transmitted data: Gaussian noise, multi-paths, Rayleigh scattering. Then, matrixes of bits are transmitted to the sink function. This function traditionally includes the generated simulation data for post-processing analysis. The sink transfers the received data to the compare function, which also receives a copy of the original data emitted by the DataSource function. This comparison module counts the number of errors and calculates the bit error rate.

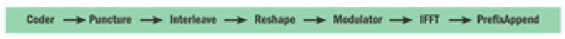


Fig 3: The transmitter module structure.

A bigger version of this graphic is available.

The transmitter module is structured as indicated on the following figure 3. * Coder: the convolutive coder introduces a redundancy into the data to be able to easily find the data emitted, even in case of perturbations on the radio channel.

* Puncture: the puncture function moderates the redundancy introduced previously by eliminating a certain number of bits.

* Interleave: the interleave function mixes the data according to a very precise algorithm to decrease the number of successive identical symbols, reducing the consumption required for the radio transmission.

* Reshape: this function distributes the bits on the frequency sub-bands which are radio frequencies available for the transmission.

* PilotInsert: the pilots are particular sequences inserted on dedicated sub-bands for improving the detection of the radio signal in reception.

* Modulator: the modulator transforms the bits into symbols: complex values (I and Q channels) processed according to a dynamically chosen pattern called constellation (BPSK, QPSK, 16QAM, 64QAM).

* IFFT: the Inverse Fast Fourier Transform function performs the conversion between the frequency domain and the temporal domain required for the output signal.

* PrefixAppend: this function performs the addition of a cyclic prefix to the obtained data.

The receiver module presents a partially symmetric structure to the transmitter module as presented in the following figure 4.

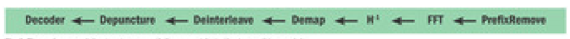


Fig 4: The reciever module structure - partially symmetric to the transmitter module.

A bigger version of this graphic is available.

The Depuncture, Deinterleave, PilotRemove, FFT, PrefixRemove functions perform the inverse operations achieved by their counterpart functions in the Transmitter.

The three following modules are more specific:

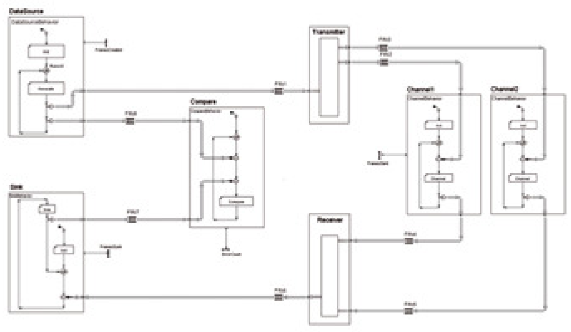
* ChannelInvert: this function performs the classical channel inversion by canceling certain effects of the air channel

based upon measures of the frequency profile of the channel.

* Demap: this function tries to find in the constellation pattern to which bit corresponds the received symbol.

* Decoder: the decoder takes the final decision concerning the value of the received bits using a Viterbi algorithm.

Testbench modeling: THALES's application model structure is derived from the functional design study detailed before. The testbench consists of two radio channel simulators (channel 1 and channel 2), a data source, a data sink and a comparator checking received data against the originally sent data - shown in the figure 5. Similarly, the receiver subsystem shows two receiver heads for simulating spatial diversity reception.

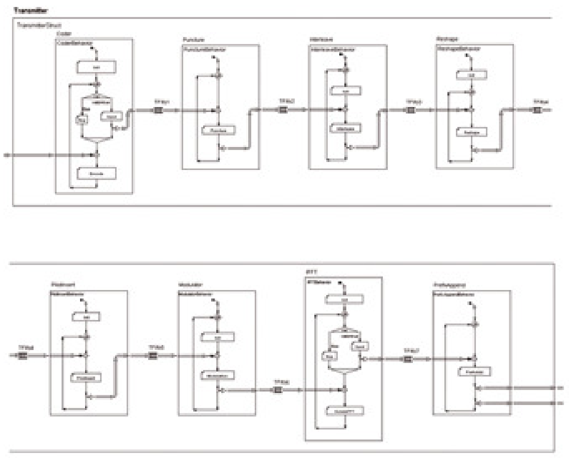


A bigger version of this graphic is available

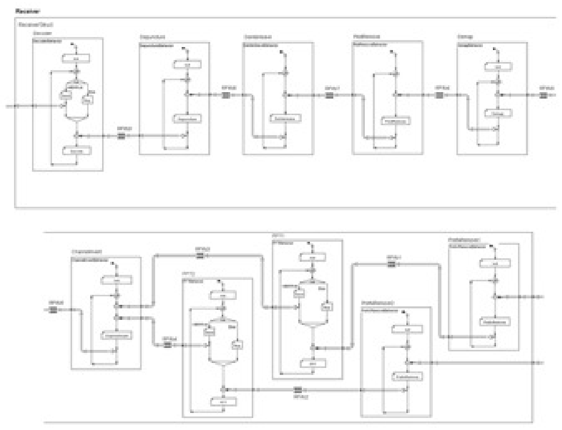
A choice is made to model all communications between modules as FIFOs, since this is typically the case, even at the hardware level. Communications between a modem system-on-chip (SoC) or board and the higher-level data sender/receiver is frequently done through a physical FIFO, in order to disconnect the real-time data transport rhythms from the upper data exploitation level. This is also true for the data sent from the emission output of the modem to the radio channel, as well as data received to the reception input of the modem from the radio channel.

A common practice is to use a FIFO on these links to limit the effect of a strictly clocked (on a bit basis) hardware transceiver part on the modem chip. Two solutions are envisaged: one uses FIFOs and the other does not. In the real world, the latter requires a more complex transceiver part.

Modem functional architecture: The models for transmitter and receiver were deduced from the internal functional organization of the solution. Figures 6 and 7 detail the transmitter and receiver subsystems.



A bigger version of this graphic is available.

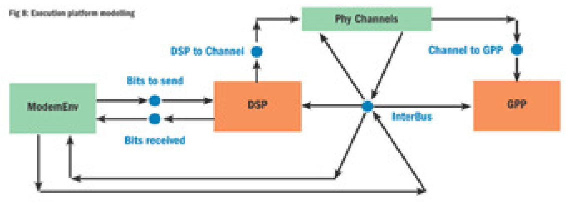


A bigger version of this graphic is available.

Execution platform modeling

In order to study the performance of the system, the identified physical architecture is described in the following figure. This executive structure is composed of two processing units: one software unit for functions having relatively high activation periods and a hardware unit to support the high-frequency activated functions.

The chosen hardware platform for the test case is dedicated to simulation and validation purposes in the analysis and design phases of the high data-rate modem. It consists of a general purpose processor (GPP) and a digital signal processor (DSP), which is to be selected later – see figure 8.



A bigger version of this graphic is available

Each processing unit is parameterized by its attributes. A link simulating the bus between the microprocessor and the ASIC is used for the hardware interconnection. The partitioning of the different 802.16 modem application components (allocation of the functions to the DSP) is done according to the performance constraints (data rate, required number of processors, power consumption).

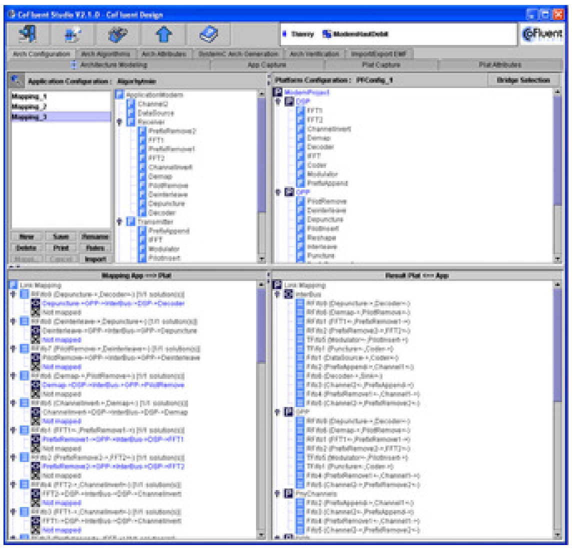
The created model has multiple possibilities for data paths, which can be individually selected to experiment with various possible hardware choices.

Three main hardware options are compared. The circulation of data frames inside the GPP or DSP is not modeled in hardware. In a completed device, it is done by passing a memory pointer from one software module to another on the same processor. This operation requires no significant time, typically less than ten processor cycles. The changes are on the modem’s interfaces to the outside world, at the data source/destination on one side, and at the radio hardware interface on the other side.

Data circulating to and from the modem and between processors uses models of hardware links. Timing information for such links is introduced in the link definition. The hardware FIFOs are defined to have a size of 3 frames, and the bus has no FIFO (size of 1) which indicates that every access to the bus must ensure it is free. If not, it must wait for the bus to become free.

None of the interfaces has concurrency (i.e. only one transfer running on every link at a given time). It is necessary to get the timing effects of bus contention, in a model where a non-trivial part of data circulation occurs through a shared bus.

Several models of the high data-rate 802.16 modem are modeled at different abstraction levels, according to the MARTES methodology. An example of a mapping of the modem function onto the platform and the distribution of functions between the DSP and GPP is shown in figure 9.



A bigger version of this graphic is available.

For high level and radio interfaces on the bus all data is exchanged via the multiple access bus on the modem. No specialized link is used in the design. Data comes from the testbench (external world) into the DSP via a bus port. It is passed back and forth between the DSP and GPP via the same internal bus with possible priorities, and then goes to the physical transmitter via the same bus.

After simulating the radio channel, the data is read back by the receiver part of the modem via the bus. It is exchanged between the DSP and GPP via the bus depending on the various algorithms to apply, and then goes to the testbench for comparison via a port on the bus.

In this case, the data source, destination, and radio interface are projected on the bus address space. This options leads to less hardware and more possibilities for the data exchanges (going either to DSP or GGP by programming). It has the drawback of an increased risk of contention on the bus and the connecting elements: all have to be aware of the timing at the frame level.

For radio interface on the bus, high level on dedicated FIFOs, in this configuration, data to send comes from the testbench to the DSP via a dedicated FIFO. It is passed back and forth between DSP and GPP via an internal bus with possible priorities, and then goes to the physical transmitter via the same bus.

After simulating the radio channel, the data is read back by the receiver part of the modem via the bus. It is exchanged between the DSP and GPP via the bus depending on the various algorithms to apply, and then goes to the testbench for comparison via a dedicated FIFO.

Circulation of data frames inside the GPP or DSP is not hardware-modeled: It is done by passing a memory pointer from one software module to another, and requires no time.

This architecture is representative of a radio interface which consists of a port projected on the bus, with the radio part implementing a hardware FIFO to put the data on the air. The FIFOs on the high-level end alleviate the real-time burden on the data generator and receiver, which need only to insure a known average throughput.

A radio interface via dedicated FIFOs configuration uses additional dedicated FIFOs on the low-level (radio) interface. There is no interference from data transfers between processors to radio interface. The FIFOs are implemented in hardware on the modem or in software on the processors via a serial interface. This simplifies the radio hardware, which can receive the data with a clock and no timing to control.

This implementation is more efficient, but is more difficult to modify. For example, deciding that it would be better to receive data via the GPP, if the chosen hardware link goes from the radio to the DSP, this would require a hardware change. In the bus version, it would be a simple software change.

A block decomposition of the modem as functional modules without algorithms is achieved first, validating that the block dispatching and description of the existing C++ modules and logical data path is retained. The circulating data remains unchanged; it is composed only of placeholders.

Global parameters allow checking a choice of data frame length and modulation types. The noise level encountered on the radio channel and the number of frames transmitted each time the model is run are also programmed by global parameters. The noise level is used only when the channel is simulated, but the frame size and symbol numbers are interpreted even at the first stage without algorithms.

Code timing parameters are introduced at the post-mapping architecture level. They may be dependent on the data size, but not on data value.

These parameters are attached to block operations (elementary computing operations inside the blocks). They can be introduced independently of the algorithms: time elapsed can be evaluated even with all blocks empty (black boxes). Even with no implementation (hardware) details and no insertion of the real code, the system's capability regarding real-time can be checked with a good precision due to this feature.

For a simple platform, such as a mono-processor integrated hardware with no complex internal data links, this has a sufficient precision. If the hardware projected is complex, the platform description seen later is a better solution.

The time used to cross inter-block links also can be simulated at the early stage, without real code running. This option allows refining the timing simulation obtained without a description of the projected hardware.

Although this can be another way to get a fast estimation of a system's capabilities, having a good precision timing description of the links inside the system means that the hardware is implicitly known. In that case, it is advisable to use the platform description described below, and determine further possibilities to compare for different implementation options.

Algorithmic simulation

At this stage, the global initial algorithm is split and distributed into the blocks. This step checks that the calculations are still correct. The data circulating in this model is representative of real data, and its integrity after crossing the whole modulation/demodulation cycle is verified.

As mentioned before, the number of symbols per sent frame and the number of OFDM symbols used are defined as generic parameter. Various selections can be simulated.

Noise can be introduced in the simulated radio channel, and errors are counted and stored for display and analysis. The model simulates multi-path propagation with two radio channels simulated as well as two receiver heads for spatial diversity exploitation.

External libraries are used by the initial C++ model: the ATLAS library, used for linear algebra functions, and the FFTW library, which implements Fast Fourier Transforms. The headers for these libraries and the binary libraries are added as external files to the CoFluent model.

A private library is introduced as a number of source files. It describes a C++ matrix data type, with most classical matrix operation (inversion, diagonalization, product, etc.). This commonly-used THALES internal code is directly copied and pasted into the algorithm description of CoFluent operations.

Also included as external source files are subroutines used for OFDM, modulation and puncture algorithms. The System C library is needed by the imported code, and is included in CoFluent Studio. A template is used by CoFluent Studio to generate a makefile for each project.

The SoC simulation stage simulates the implementation of the modem in a two-processor hardware architecture, loosely derived from the TI OMAP 5910-5912.

The testbench, composed of a random data generator, two radio channel simulations and sent/received data comparison, is described as hardware. It reflects the environment of a real modem. The various software elements have been given execution times to allow the following performance tests. If the projected hardware is complex (multiprocessor, intelligent data links, etc.) it is necessary to use a different set of values according to the projection of the architecture on hardware parts. An operation executes at a different pace based on the processor running it.

Once the platform is described, the links between blocks are known. This is the critical part where it is necessary to use previously recorded performance data for the diverse functional blocks and for the links. Accurate timing data entered here is the obvious key for a useful simulation in the next step.

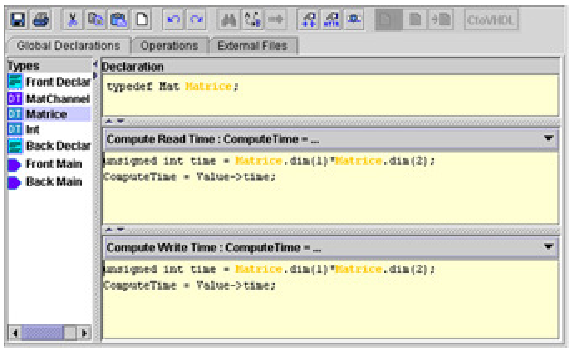
During the evaluation, the modem was mapped in three different ways to the architecture: * Data enters through FIFOs towards the DSP, is exchanged between DSP and GGP via the internal bus, and then is sent to the radio via the same bus. This was the first tested configuration. * The data goes from the general purpose processor to radio via another set of FIFOs, and software blocks are dispatched differentlybetween DSP and GPP. This test shows that changing the dispatching of elementary operations can get better results in term of throughput. * All data exchanges go through the inter-processor bus. This is the simplest, in terms of hardware, configuration. It puts a higher burden on the bus timings and is likely to increase the complexity of neighboring hardware for the time-shift of data frames due to bus contentions.

To simulate the data path delays, the user defines the delay for every data transfer in the simulated system.

For standard data types (byte, int, float …) the size of the data is known and the transfer duration can be defined from it: the DATASIZE macro allows calculating the number of bytes circulating through an interface. The delay is a function of it. For user-defined data types, the interfaces in the model transport a pointer to the object transported. The USERDATASIZE macro can be used to obtain the value of a user-defined field representing the data size.

As the physical interfaces transport the full data and not a pointer, CoFluent Studio has a mechanism to calculate the real delay introduced. It uses a macro called FCTCOMPUTETIME. This calls a user-defined function to get the time required for the transfer.

The pointer matrix object used in the modem is made up using a two-dimensional size and contains exclusively double elements. From these characteristics, the function for FCTCOMPUTETIME is written and introduced as seen in the following figure.



```
typedef Mat Matrice;
```

```
unsigned int time = Matrice.dim(1)*Matrice.dim(2);
ComputeTime = Value->time;
```

```
unsigned int time = Matrice.dim(1)*Matrice.dim(2);
ComputeTime = Value->time;
```

A bigger version of this graphic is available.

Using the above elements, the read/write times for a link transporting matrix objects can be defined as functions of the matrix size, as it's the case in a real system.

Varying the parameters of the model and changing between different block dispatching and data path possibilities clearly shows the executable model created with CoFluent Studio is an efficient aid in determining both hardware/software tradeoff and module distribution between processors.

Although initial training is highly recommended to maximize the benefits of the new methodology, most design teams find the tool beneficial within a matter of weeks. The tool is efficient in comparing hardware/software tradeoffs and board or SoC design options. The level of confidence expected depends mainly on the quality of the performance data the user inputs. CoFluent Studio in itself allows a large range of possibilities to define the performance data.

Here's what THALES wrote in their report as a conclusion of their work with CoFluent Studio: "CoFluent Studio gives access to a number of parameters. This gives it sufficient precision to be an efficient tool in predicting the capacity of an architecture/platform couple. The precision of the results delivered depends almost only from the quality of the timing data put into the model.

"If this tool is to become mainstream in an organization, it will require to carefully gather timing information on what performance is attained by every algorithm/processor couple used. An experimental data library should be created and maintained in order to allow this.

"This data can then be introduced in a projected system. With only a limited expertise, capitalized knowledge will then allow to have a quick and accurate answer to the question whether this platform is fast enough to support this system or not. Carefully putting feedback on resulting implementations into the performance database will increase its accuracy over time."

About the authors

Thierry Saunier is head of the modelling engineering laboratory at THALES.

Vincent Perrier is co-founder and director of CoFluent Design.

This story appeared in the February 2009 print edition of *EE Times Europe*European residents who wish to receive regular copies of EE Times Europe, subscribe here.

See other stories from this issue here.

You can download a digital edition of the latest *EE Times Europe* print edition here.