Prototyping of the Receiver Unit for a Broadband Access Network

A. Hein, J. Dalcolmo, P. Le Corre, R. Lauwereins, M. Adé
Kard. Mercierlaan 94
K.U. Leuven
ESAT/ACCA Laboratory
B-3001 Heverlee, Belgium
email: Axel.Hein@esat.kuleuven.ac.be

Abstract

To exploit the potential of two-way communication on CATV networks for advanced interactive telecommunication applications like video-on-demand, a high speed modem for up-stream communication is currently under development. We are using the rapid prototyping tool GRAPE to evaluate the digital receiver part of the modem which is described in detail in this paper. The sampling rates achieved for a 16-QAM modem on multiple DSP processors are reported.

1. Introduction

Recently, coaxial cable networks have received much attention in the context of interactive applications [3, 5]. In countries with a high penetration of CATV (in Belgium more than 90%), it may serve as an alternative to classical telephone networks. The envisioned applications are telephony, interactive television, home-shopping, video-on-demand, high-speed Web browsing, etc [7]. Since these applications require the possibility of interactive use, two-way communication has to be provided on the networks which initially have been designed for only a one-way broadcasting of television signals.

What is aimed at now, is a 10 Mbit/s upstream link (from the subscriber to the head-end station) and a downstream link (from the head-end station to the local subscribers) with higher bit rate. Especially the upstream is challenging because very little is known about the upstream channel, and communication standards are still under development [4]. The projected frequency band for upstream communication is in the 5-25 MHz range.

Since the basic configuration of the network is hybrid fibre-coax, the upstream signal will first travel through coax before entering a fibre node and going through the fibre trunk to the head-end station. The coaxial part will bring along some serious channel impairments which have to be compensated for at the receiver. The most important impairments are mentioned below without going into too much detail. Group delay distortion (ie signals at different frequencies propagating with a different velocity) causes severe inter-symbol interference at the receiver. Micro-reflections are caused by discontinuities in the transmission medium and cause part of the signal energy to be reflected. Ingress noise models the interference caused by the antenna-like properties of the cable. Burst noise typically originates from household appliances such as electrical motors. Besides this there are common path distortion products, thermal noise, impulse noise, non-linearities, phase-noise frequency offset etc. Adding to this the variations in the network stemming from the variability of the number of trunk, bridge and distribution amplifiers it becomes quite clear that it is very hard to build a channel model which incorporates all these statistical and non-statistical phenomena observed in real networks. Studies [5] have shown that due to this inhospitable environment, less than half of the spectrum will be available at any given time instant. The absence of a good mathematical channel model necessitates the *real-time prototyping* of the complete set-up, including the cable, after off-line simulation and before commitment to silicon is made. Real-time prototyping has several benefits compared to off-line simulation:

- It enables algorithm verification on the real channel.
- It allows for more extensive testing under more varying conditions: 24 hours a day measurements

are possible as well as tests with transmitters placed on different locations.

- It gives faster feedback when modifying algorithmic settings: the prototype allows to modify algorithmic parameters on the fly, without recompilation and to view its effects in real-time on the next received data packet.

We aim at developing a 16-QAM modem for the upstream communication channel described above. The projected bit rate is 10 Mbit/s at an overall bit-error rate of 10^{-10} including software-based error correction. The transmission payload consists of ATM cells and the multiple access protocol is TDMA.

A prototype of the receiver part of a 16-QAM modem is currently under development at the ESAT/ACCA research group of the K.U. Leuven. A sketch of the modem architecture is shown in Figure 1. The algorithm for the decoding of the data has been simulated and is to be implemented in an ASIC. It is desirable to test the algorithm on data on a real network, and to measure bit and frame error rates. However, workstation simulation of the algorithm is in the range of 10^4 slower than real time operation and therefore too slow for such an evaluation. It is rather obvious, that fast execution time of the prototype is mandatory; this can be done by porting the algorithm to a set of DSP processors or FPGAs. Since DSPs provide larger flexibility, the algorithm of the receiving unit has been ported to a set of DSPs, working in parallel, in order to achieve suitable performance. GRAPE has been chosen as the rapid prototyping tool to implement the algorithm on a set of DSPs, because it allows automatic task assignment, routing, and scheduling, gives rapid estimation of the performance, and automatically generates the code required for the DSPs [6].

Since the software running on a set of DSPs is still not capable to provide real time data throughput, the problem had to be divided into a software and hardware domain. The software runs the actual receiver algorithm which is to be tested, the hardware provides the access to real time sampling rates, at least for the duration of one or a few data bursts, ie data for a single network user can be processed in real time.

In the following sections we present the components of the receiver unit of the modem – the Sample Rate Reducer (section 2) and the processing unit of the receiver (section 3). Measurements are presented in section 4, and the main aspects of the paper are summarized in the final section 5.

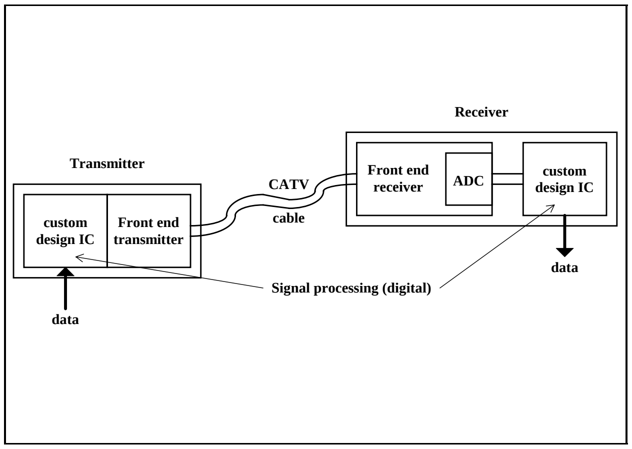

Figure 1. Modem Setup

2. Sample Rate Reducer

The modem algorithm running on a set of DSPs cannot keep up with the rate of incoming data in real time. However, in order to test the algorithm on a real network, real time samples have to be captured and played back slower to the DSPs running the algorithm under test. This is the task of the Sample Rate Reducer shown in Figure 2. This device, developed by the ESAT/ACCA group at K.U. Leuven, captures a frame of real time data, stores it, and plays it back to the DSP boards. The number of samples per frame is programmable, and the Sample Rate Reducer has features to facilitate debugging.

Furthermore, the hardware has the task of synchronization with the data bursts: The energy level on the net is calculated and a discriminator triggers on the beginning of a data burst. Since this must happen in real time, it could not be implemented in software. Synchronization is needed to avoid storage and software processing of long periods of samples between data bursts. Elimination of these periods is what actually provides the reduction of the sample rate needed for software emulation of the receiver algorithm.

2.1. Sample Rate Reducer Concept

The main function of the Sample Rate Reducer is that of a frame grabber. In order to test the modem receiver algorithm on the DSPs, samples with predefined content are sent by a test transmitter. The data arrives from the receiver ADC at a sampling rate of approximately 10 MHz and is then stored in two circular buffers simultaneously. One of these is used to buffer data flowing to the DSPs, the other one is buffering a

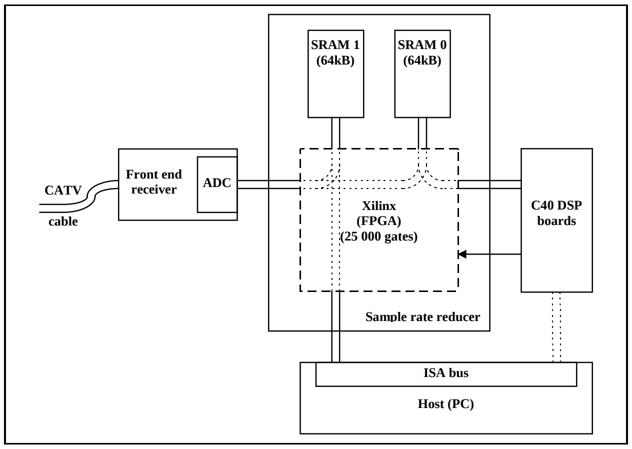

Figure 2. Sample Rate Reducer

copy for debugging purposes and allows downloading of the incoming data to the host PC simultaneously with new data acquisition. When a frame of data has been detected, the data acquisition is temporarily suspended, and one copy of the data, including pre-trigger data is transferred to the DSPs. The DSPs run the modem algorithm on it, then compare it with the expected data content and return at the completion a flag indicating success or failure of the reception.

If the flag returned by the DSPs indicates an error, the second copy of the data is moved to the host computer via an ISA bus interface, so it can be processed off line by feeding it back into simulation to improve the receiver algorithm.

Thus, the better observability of simulation models is combined with the higher speed and 24-hours-a-day unattended measurement capabilities of prototyping.

Even while a frame is being downloaded to the host, the Sample Rate Reducer restarts with collection of new data, now only saving one copy of incoming data, until the host download is completed. In this fashion, there is no interruption in the data flow, even when debugging information is being dumped to the host computer. Naturally it is not possible in this way, to provide debugging information on a continuous basis, but only for the first of a set of erroneously received frames.

The net result is that the output data rate is in the order of about 200 times slower than the incoming rate, when using a single DSP, which means that one out of 200 frames could be analyzed, if the incoming data stream would be near continuous. The actual rate is dependent on the number of DSPs used to run the receiver algorithm. For test purposes, the repetition rate of the frames on the network is reduced by a factor of 200, but the frames themselves are sent with real time speed.

2.2. Implementation Details

Unfortunately it was not possible to use the rapid prototyping environment GRAPE to implement the data flow through the FPGA, as has been done for the DSP part, because the necessary VHDL code generator within GRAPE is still under development. Therefore we had to resort to code the FPGA by hand, using VHDL.

All the logic on the board of the Sample Rate Reducer, except for an ISA Bus address decoder, is implemented in a Xilinx FPGA chip. All the logic on this chip was designed in VHDL.

Besides the 16 bit ISA bus interface, there are TMS320C40 style DSP links implemented, to communicate with the DSPs, and a general purpose parallel interface to capture modem data coming from an ADC. All I/O interfaces are electrically buffered. The FPGA has direct simultaneous access to two 16 bit wide, 64 kWord deep static RAM chips, with 15ns cycle time.

The data flow is from the parallel interface through the FPGA to the two RAMs, then from one RAM via the FPGA to a link to the DSP boards, and for the debugging information from the second RAM via the FPGA to the ISA bus. Various parameters like the frame length and repetition rate are programmable via the ISA bus interface.

2.3. FPGA

The FPGA is in circuit re-programmable via a standard Xilinx XChecker interface, and can also be probed via this interface. In addition there are a few facilities on board to ease debugging of the board or any design inside the FPGA.

The design programmed into the FPGA is modular and consists of several blocks shown in Figure 3.

- **Energy Calculation and Burst Detection**: The signal energy on the network cable can be calculated by $I^2 + Q^2$, where I and Q are a pair of successive ADC samples. The energy calculation is implemented as an approximation to avoid the high hardware cost of multipliers. The energy calculated from the pairs of samples is averaged over four samples and compared with a threshold. If this average is above theshold four times consecutively, it is assumed to be the beginning of a data burst. Since the 12-bit wide samples arrive at a

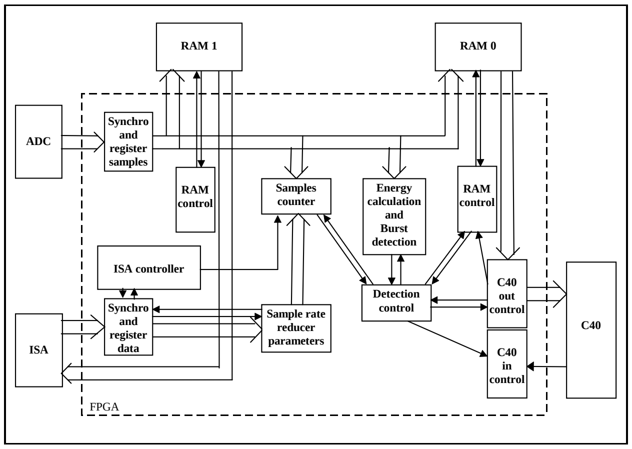
RAM 1
RAM 0
ADC
Synchro and register samples
RAM control
Samples counter
Energy calculation and Burst detection
RAM control
ISA controller
Detection control
C40 out control
C40
Synchro and register data
Sample rate reducer parameters
ISA
C40 in control
FPGA

Figure 3. Overview of the Receiver Component

rate of 10 MHz, the speed of the energy calculation has to be taken into account too. The various operations that take place are pipelined in order to achieve the required throughput.

- **ISA Controller**: A PC interface to a set of parameter registers and to read status flags as well as captured data frames for debugging.
- **Sample Rate Reducer Parameters**: A set of registers for: the threshold value for burst detection, the size of the data frame, the trigger position, the time to wait for the DSPs etc.
- **Sample Counter**: A count down counter decrementing at each incoming sample advances the state machine when it reaches 0. It is used to position the trigger within the data, to stop and start data aqcuisition and to time the waiting period for the DSPs.
- **Detection Control**: This is the top level state-machine of the Sample Rate Reducer, and driven mainly by the state of the Sample Counter, the input from the Burst Detection and the DSPs.
- **RAM Controllers**: Essentially silicon tape recorders, storing the incoming data at high speed to replay them slowly.
- **C40 Controllers**: The DSP word length is 32 bits which the link sends asynchronously in four times 8 bits. Both blocks handle the hand-shaking signals and convert the data format (the FPGA uses a 12-bits two's complement representation of the data).
- **Synchro and Register**: The ISA bus is running asynchronously with the FPGA, and there is an unknown difference in phase between the ADC and the FPGA. These blocks are used for synchronization.

3. Processing Unit of the Receiver

The processing unit of the receiver is responsible for the decoding of the incoming symbols and for the post-processing which includes checking for bit errors and notifying the sample rate reducer in the case of errors.

The tasks of the processing unit as well as their interdependencies are defined within the GRAPE environment as an extended data-flow model which is called *cyclo-static data-flow* (CSDF) [2] and represents an extension of Lee's *synchronous data-flow* (SDF) [8]. CSDF maintains all the properties of SDF but allows for the consideration of *cyclically changing behavior* such as time-multiplexed channels with a static and cyclically repeated period. Although this cyclo-static behavior is known at compile time and is independent of any run-time data, it cannot be specified within the SDF framework which requires fixed and constant token productions and consumptions.

GRAPE's design flow consists of several phases [1] as shown in Figure 4. Regarding the data-flow representation, the application has to be set up as a directed graph *G=(N, E)*, where the nodes *N* represent computation tasks, and the edges *E* depict the communication of the results (commonly called the *tokens*) between a producing and a consuming task. The functionality of the nodes is specified in a conventional high-level language like C (for DSP processors) or VHDL (for FPGAs). The number of tokens a task consumes and produces during an execution phase of a task is known at compile time, allowing for a compile-time analysis of the graph in the next phases of GRAPE's design flow and leading to highly efficient run-time code. GRAPE embeds the task descriptions in shells which handle all the communication.

The target hardware is represented as a connectivity graph with an indication of the amount and type of resources each processing device possesses. In the following step of GRAPE's design flow, the amount of resources required by each of the tasks when executed on each of the processing devices, is estimated. Then, the application is mapped onto the target hardware, ie the assignment, routing, and scheduling of the tasks are automatically optimized to minimize the buffer lengths and the total execution time. The final code is generated depending on the target hardware, an estimation of the execution times is given, and the application can be downloaded on the target hardware. It should be emphasized that the target hardware can be changed rapidly without modifying the data-flow model of the application and vice-versa. This independence of application and target hardware has tremendously facilitated the performance measurements of the application on different hardware platforms as will be presented in section 4.

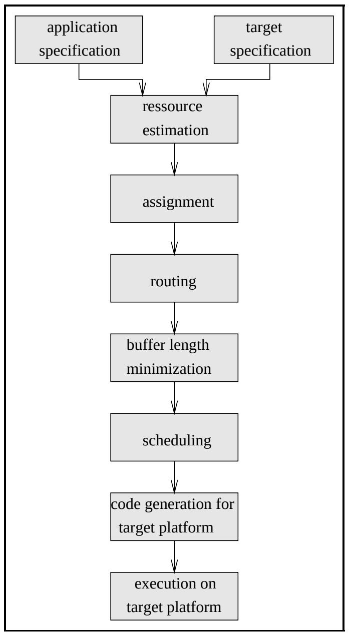
application specification
target specification
ressource estimation
assignment
routing
buffer length minimization
scheduling
code generation for target platform
execution on target platform

Figure 4. Flow of GRAPE Design Process

The architecture of the 16-QAM receiver unit which has been designed by IMEC and the ESAT/SISTA research group consists of the following tasks:

- **DMUX** (Demultiplexer) : The 4-phase demultiplexer scales the incoming values before they are forwarded.
- **FIL[IQ]** (FIR Filter) : These two FIR filters downsample and filter the incoming data by performing a convolution with pre-computed parameters.
- **AGC** (Automatic Gain Control) : The AGC controls and corrects the gain. The changes between the internal idle and active states depend on the input data and state parameters.
- **EQ** (Burst Acquisition Equalizer) : The EQ task performs the detection of symbol sequences. The state changes depend on the input data and state parameters.
- **LMS** (Symbol-Spaced Adaptive Equalizer) : Equalizing is performed based on feedforward and feedback convolutions.
- **DEMAP** (Demapper) : The constellations of 16-QAM or 4-QPSK, respectively, are mapped back to symbols. The 4-QPSK encoding is provided as an alternative to 16-QAM but is not further taken into account within this paper.

Figure 5 gives an overview of the receiver unit represented as a data-flow model.

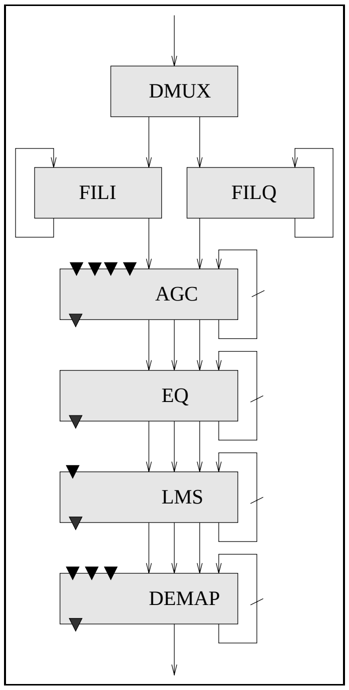

Figure 5. Data-Flow Model of the Application

During each static schedule DMUX is executed four times, and all the other tasks are triggered twice, ie the incoming data are downsampled. In each of the four executions of DMUX another set of operations is started. The order of these four phases is known at compile-time and can be statically assigned and scheduled; the consideration of this cyclo-static behavior (CSDF) is one of the outstanding characteristics of the GRAPE tool.

The execution of LMS and DEMAP is data-dependent, ie they are triggered by EQ. LMS and DEMAP perform operations only when *useful* data is coming in, and in that case only once per schedule; therefore, EQ downsamples by a factor of two. A static assignment tool as it is used by GRAPE cannot handle this data-dependent execution which can be considered as dynamic data-flow. Hence, LMS and DEMAP are triggered twice a schedule, such as AGC and EQ, but the incoming data provides information whether operations have to be performed at all.

4. Performance Measurements

The original code of the application has been optimized to speed up the execution times on the DSP processors. This step clearly increased the sampling rate of the application. Table 1 shows the sampling rates for simulation on a SUN UltraSparc Workstation and for one, two and four DSP processors TMS320C40 before (T_{no-opt}) and after (T_{opt}) code optimization, which includes elimination of redundant operations, loop unrolling, and introduction of additional variables to avoid address calculations. Moreover, expensive `switch` conditions have been replaced by nested `if` statements.

Architecture	T_{no-opt} [kHz]	T_{opt} [kHz]
Simulation	0.19	–
1 C40	12.3	22.7
2 C40	23.6	42.4
4 C40	23.6	70.4

Table 1. Sampling Rates of the Processing Unit

When moving from 2 DSPs to 4 DSPs for the unoptimized algorithm, no improvement of the execution time can be achieved. This is due to the fact that the AGC task takes about the same execution time per schedule as all the other tasks together; in its original version, the shifting of numerous state parameters is performed by copying them within arrays. By replacing this expensive mechanism with a few additional variables pointing to the *logical* begin of the arrays, the copy operations can be avoided and the execution time of the AGC was reduced from 1624 down to 356 cycles.

Several *runtime-* and *back-parameters* have been added to the application. Runtime-parameters make it possible to modify algorithmic parameters at runtime, whereas back-parameters provide instantaneous feedback from the application running on the DSPs to the user or a control program running on the PC (in Figure 5 the runtime-parameters are drawn as black triangles pointing to the corresponding block, and the back-parameters are represented as dark-grey triangles pointing out of the block). The automatic gain control (AGC) can be bypassed to set the gain value manually, and threshold values may be modified at runtime. Also, the modulation type (4-QPSK or 16-QAM) can be set. Additionally, a derandomizer can be activated and initialized (DEMAP). The implemented back-parameters provide information about the gain value (AGC), the tap coefficients (EQ and LMS), and the correctness or error of the received and decoded samples (DEMAP).

The runtime- and back-parameters allow for testing and monitoring of the application. When problems are encountered, these values are used to improve the application by going back to more flexible but slower simulation runs.

5. Summary

Although continuous real-time sampling rates cannot be achieved, prototyping is shown to be a needed and useful means to evaluate the interaction of such a modem design with the real channel. The use of an advanced environment like GRAPE in combination with programmable hardware strongly encourages the fast prototyping and makes it hardly more expensive than simulation.

GRAPE's advanced use of run-time readable and writable parameters facilitates the monitoring, debugging, and fine-tuning of applications assigned to hardware platforms which can be exchanged easily and independently from the application. By using DSPs instead of workstation simulation the execution times of the algorithm are tremendously speeded up by a factor in the order of 10^2.

6. Acknowledgments

R. Lauwereins is a Senior Research Associate with the FWO. M. Adé is a postdoctoral researcher of the K.U. Leuven-DOC. J. Dalcolmo and A. Hein are postdoctoral researchers, P. Le Corre is a master of engineering at the K.U. Leuven. Research has been supported by Siemens Atea and the Flemish Government via the Flemish Institute for the Advancement of Scientific-Technological Research in Industry (IWT). This project has been partly made possible by an FWO Krediet aan Navorsers and is partly sponsored under Texas Instruments' Elite University Program. K.U. Leuven-ESAT is a member of the DSP Valley network.

References

[1] M. Adé, R. Lauwereins, and J. Peperstraete. Hardware-software co-design with GRAPE. In *Proc. 6th Int. Workshop on Rapid System Prototyping*, pages 40–47, June 1995. Chapel Hill, NC, USA.

[2] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. Cyclo-static dataflow. *IEEE Transactions on Signal Processing*, 44(2):397–408, February 1996.

[3] R. Comerford and S. Tekla. Wired for interactivity. *IEEE Spectrum*, pages 21–28, April 1996.

[4] W. End. IEEE project 802.14 : Standards for digital convergence. *IEEE Communications Magazine*, pages 20–23, May 1995.

[5] L. Goldberg. Cable modems: The journey from hype to hardware. *Electronic Design*, pages 65–80, April 1996.

[6] R. Lauwereins, M. Adé, M. Engels, and J. Peperstraete. GRAPE-II: A system-level prototyping environment for DSP applications. *IEEE Computer*, 28(2):35–43, February 1995.

[7] R. Lauwereins, M. Adé, P. Vandaele, M. Moonen, and P. Schaumont. Prototyping quadrature amplitude modulation for two-way communication on CATV networks. In *Proc. of 7th Int. Conference on Signal Processing Applications and Technology ICSPAT*, pages 1570–1574, 1996. Boston.

[8] E. Lee and D. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, C-36(1):24–35, January 1987.