J Sign Process Syst (2013) 73:301–314
DOI 10.1007/s11265-013-0762-2

Physical Layer Development Framework for OsmocomBB

Harald Kröll · Stefan Zwicky · Benjamin Weber · Christian Benkeser · Qiuting Huang

Received: 26 October 2012 / Revised: 1 February 2013 / Accepted: 24 April 2013 / Published online: 29 May 2013

Abstract The open source GSM protocol stack of the OsmocomBB project offers a versatile development environment regarding the data link and network layer. There is no solution available for developing physical layer baseband algorithms in combination with the data link and network layer. In this paper, a baseband development framework architecture with a suitable interface to the protocol stack of OsmocomBB is presented. With the proposed framework, a complete GSM protocol stack can be run and baseband algorithms can be evaluated in a closed system. It closes the gap between physical layer signal processing implementations in Matlab and the upper layers of the OsmocomBB GSM protocol stack. An embedded version of the system has been realized with FPGA and PowerPC to enable real-time operation. The functionality of the system has been verified with a testbed comprising an OpenBTS base-station emulator, a receiver board with RF transceiver and our developed physical layer signal processing system.

This work has been presented at the SDR 2012 Wireless Innovation Forum Europe conference [1]. The open-source MatPHY framework is licensed under the GPLv3 license and can be downloaded at: http://code.google.com/p/matphy.

H. Kröll (✉) · S. Zwicky · B. Weber · C. Benkeser · Q. Huang
ETH Zurich, Integrated Systems Laboratory,
Gloriastrasse 35, 8003, Zurich Switzerland
e-mail: kroell@iis.ee.ethz.ch

S. Zwicky
e-mail: zwicky@iis.ee.ethz.ch

B. Weber
e-mail: weberbe@iis.ee.ethz.ch

C. Benkeser
e-mail: benkeser@ruag.com

Q. Huang
e-mail: huang@iis.ee.ethz.ch

Keywords Baseband signal processing · Physical layer hardware architectures · OsmocomBB · GSM protocol stack · L1CTL messages

1 Introduction

Recently, the open source community has discovered the GSM protocol as an interesting exploration area, mainly for security aspects. Among the various successful attempts to implement open source software for several parts of the GSM network, the community behind the Open Source Mobile Communication Baseband (OsmocomBB [2]) project offers implementations in C of the data link layer (L2) and the network layer (L3) of the mobile station (MS). These can be run on various MSs together with the physical layer (L1), which is partially implemented on a Digital Signal Processor (DSP), and partially in dedicated hardware. To this end, the L1 source code running on the DSP and the corresponding application interface (API) have been analyzed by reverse engineering of chipsets used in specific legacy phones. Thus, OsmocomBB is able to provide L1 source code cross-compiled for several DSPs. In this way the L1 realization can be executed directly on MSs. Lower-level parts of the L1 which are implemented in dedicated hardware can be accessed via the DSP's API.

Unfortunately, it is very difficult to get insight to layers below the API, which limits the scope of new applications and implementations. Crucial tasks of the digital baseband domain, such as channel equalization or decoding, are mostly implemented as accelerators in dedicated hardware. Therefore, they cannot be further investigated. This deficiency exacerbates OsmocomBB to be used for (research) activities on the physical layer, which includes analog and

digital front-end, baseband signal processing and L1 control functionality.

The signal processing, hardware development, and communication technology communities have strong interest in an expandable baseband development framework with an interface to L2 and higher layers of the GSM protocol stack. OsmocomBB's L1CTL protocol between L1 and L2 is well defined, but there is no development environment available in an ubiquitous scientific computing language, such as Matlab or GNU Octave, which can be connected to L1CTL. A framework with an interface of this type simplifies the validation of the functionality of baseband implementations towards higher layers in a compact system without expensive measurement equipment. Baseband engineers can use OsmocomBB during the design process and during testing of signal processing blocks that require interaction with L2/L3.

Contribution: in this paper, a Matlab-based physical layer development framework architecture with an appropriate interface to the L2/L3 implementation of OsmocomBB is presented. The framework contains digital baseband processing with corresponding L1 controller and time processing unit (TPU), as required for GSM receivers. The baseband processing is partitioned into a digital front-end, detector and decoder unit. Each unit comprises a controller and so called *signal processing primitives*, which carry out essential tasks like filtering, symbol detection, parameter estimation, bit scrambling, and decoding. Functional verification of the implemented framework and interface is performed by processing recorded samples from a base transceiver station (BTS). In addition, in order to enable real-time operation, an embedded version of OsmocomBB was built and a VHDL implementation of the framework was developed and mapped to FPGA. The embedded versions of L1 to L3 were executed on a testbed with a state of the art RF transceiver, receiving samples over the air interface from a BTS.

Outline: The paper is organized as follows. In Section 2 an overview on mobile phone architecture is given, and the need for crossing the boundary between L1 and L2/L3 is substantiated. Fundamentals of the GSM air interface between BTS and MS are explained in Section 3. An interface that connects OsmocomBB with a physical layer Matlab implementation is presented in Section 4. The Matlab framework architecture and its operation is explained in Section 5. Section 6 explains how the framework is mapped to an FPGA and how an embedded version of L2,L3 was built. The testbed setup is described in Section 6.2 and Section 7 concludes the paper.

2 The Missing Link

The protocol stack for GSM is a layered architecture based on the concepts of the ISO Open Systems Interconnection (OSI) model with 7 abstraction layers. The layered structure allows the distribution of work to specialists that can focus on a specific layer without having to consider the multitude of problems and issues that occur in the remaining 6 layers. In particular, baseband signal processing algorithms and architectures for the physical layer can be developed by neglecting L2/L3 procedures or operations of even higher layers. The separation of layers in the GSM standard has led to the classical partitioning of hardware in mobile phones, as depicted in Fig. 1.

Digital baseband signal processing tasks with low and medium computational complexity are typically executed on the *Baseband Processor*, a power-efficient DSP optimized for mobile applications. The most complex parts of the digital baseband signal processing are usually directly mapped to dedicated hardware accelerator blocks, in order to achieve the required performance (e.g., bit error rate, throughput) at reasonably low power consumption. Instead, L2 procedures are suitable for an integration in software, because computational complexity is fairly low and high flexibility is required. Therefore, these tasks are typically realized on a reduced instruction set microprocessor (RISC), the *System Processor*, which is connected to the accelerators via the DSP's API.

Although the strict separation of layers simplifies independent development and integration of the specific layers, it impedes optimizations and applications that require crossing the layer boundaries. For example, hybrid automatic repeat request (HARQ) which is a key feature of modern mobile communication standards to enable high average throughput, requires interaction between L2 and channel decoding in the physical layer. The HARQ technique specified for GSM/EDGE [4] is called *Incremental Redundancy* (IR). IR manages the storage of erroneously received data packets and the combination of the stored data with retransmissions of the same data packet. The combination of the received data packet with previously received and stored data significantly increases the probability of correct decoding.[1] Therefore the average data throughput is increased. Channel decoding is a computationally expensive baseband signal processing task in the physical layer, whereas the organization and controlling of re-transmissions, and the memory management of the stored data blocks is a procedure, that is typically controlled by L2/L3 layers (see for example [3]). Therefore, in order to simulate the entire IR functionality, in order to evaluate average receiver throughput (with IR enabled) accurately, and in order to optimize IR implementations, being able to operate across the layer boundary between physical layer and upper layers is desired from a designer's point of view.

[1] Different puncturing schemes are usually used, in order to increase the information gain with each re-transmission. Refer to, e.g., [5] for further details.

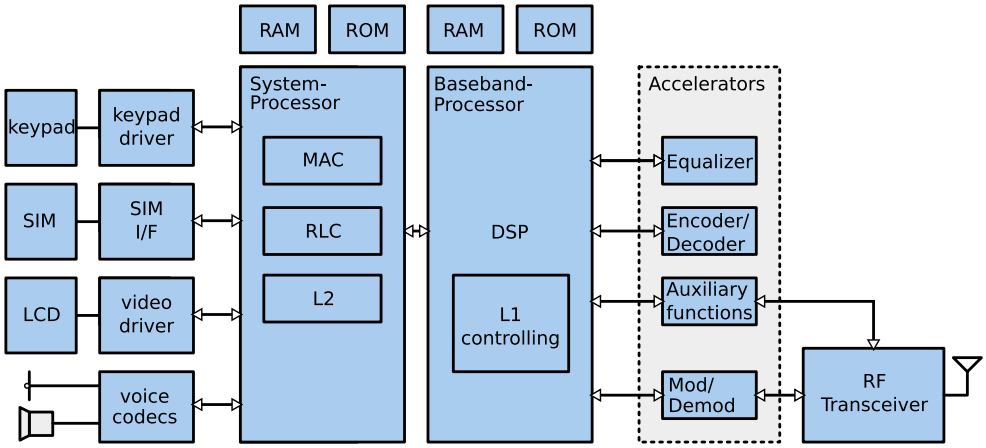


Figure 1 A common GSM MS architecture. The physical layer is distributed over a DSP and hardware accelerators [3].

More than that, having access from higher layers to the physical layer and vice versa renders new applications possible. Physical layer procedures, intermediate results of baseband signal processing blocks, or simply raw baseband samples can be monitored from higher layers, which simplifies debugging and enables new visualization opportunities of physical layer operations. An example exploiting layer crossing is *user cooperation*. In an exemplary user cooperation scenario, several mobile devices M_i support a mobile device M_0 by acting as its remote antennas. Raw received baseband samples of the mobile devices M_i are forwarded from their physical layer to their application layer. From there, a mobile application (*app*) organizes the transmission of these samples to M_0 via an ad-hoc radio technology, where the samples are combined in the physical layer. Various combining schemes (e.g., [6]) can be applied in order to improve the probability of correct decoding.

We conclude that there is a need to have access to the physical layer in mobile phones and to be able to model physical layer functions in combination with higher layers. In the following section we describe our approach of interfacing OsmocomBB with a physical layer development framework.

3 The GSM *Um* Air Interface

The *Um* air interface is defined as the air interface between a MS and a BTS. It is based on a TDMA frame structure where each frame is subdivided into 8 timeslots, each lasting about 576.92 μs. A timeslot carries a GSM burst containing 156.25 symbols.[2]

Five burst types with different configurations are specified for GSM. Three of them are relevant for this work and will be explained next. A normal burst (NB) consists of two information blocks of 58 bits and a midamble (training sequence) of 26 bits, which is known at the receiver such that it can be used for channel estimation. The frequency correction burst (FB) contains 142 logical 'zero' bits, which result in a complex sinusoid due to the nature of the GMSK modulation. The main scope of the FB is to enable the MS to synchronize to the carrier frequency. The synchronization burst (SB) carries a 64 bit midamble for synchronization in time and channel estimation, and 78 data bits, which provide the base station identification code (BSIC) and other system information (SI). Each MS in a cell can adjust its internal timebase to the timebase of the BTS by searching for the *beacon carrier*, which carries the *broadcast control channel* (BCCH). The beacon carrier is the carrier with highest power transmitted from a BTS. The beacon carrier broadcasts SI messages on radio blocks which consist of four NBs, as well as FBs and SBs in a repeating pattern. Thus, FB and SB are broadcast approximately every 47 ms on the beacon carrier.

According to the GSM specifications, a MS shall keep track of the time base by maintaining four counters in order to guarantee correct receive and transmit timing. The quarter symbol (QN) counter ranges from 0 to 624. The bit number counter (BN) counts the bits of a burst (0 to 156) and the timeslot counter (TN) counts the timeslots (0 to 7) within a frame. The frame number counter (FN) reaches from 0 to $2048 \cdot 52 \cdot 26 - 1$ (the number of frames in a *hyper frame*). For more details on the GSM Um air interface refer to [7].

4 *Phyconnect:* A New Interface for the L1CTL Protocol between L1 and L2

The GSM specifications do not foresee a detailed protocol for the communication between L1 and L2. The GSM

[2]In basic GSM only GMSK modulation is supported, where symbol is equal to bit.

Table 1 L1CTL message examples. Note that in the OsmocomBB sourcecode the messages are denoted with a L1CTL_ prefix.

Functionality	L1CTL messages
Reset PHY	RESET_REQ
	RESET_CONF
Synchronization	FBSB_REQ
	FBSB_CONF
Power Measurement	PM_REQ
	PM_CONF
Control Channel Mode	CCCH_MODE_REQ
	CCCH_MODE_CONF
Data indication	DATA_IND

standard [8] defines basic messages[3] for the communication with L2. They are subdivided into request (REQ), confirm (CONF) and indication (IND) message types. The messages of OsmocomBB's L1CTL protocol are inspired from these message types of the GSM standard. A set of examples for L1CTL messages is given in Table 1.

The default OsmocomBB interface implementation between L1 and L2, called *osmocon*, uses a serial link with HDLC protocol [9] to load the cross-compiled software into the phone's DSP memory. Using L1CTL messages, this software (often called *firmware*) communicates via *osmocon* with a Unix domain socket for the connection to L2/L3 running on a host computer.

In an attempt of replacing the DSP's firmware with a physical layer Matlab implementation, the Unix domain socket needs to be connected to Matlab. Unfortunately, Matlab does not directly support Unix domain sockets. Therefore, an interface written in C is required for the socket communication. One solution would be to have such an interface embedded in a MEX function, such that it could be called inside a Matlab script. Alternatively, the interface could execute Matlab commands directly by means of Matlab engine function calls. However, both MEX function calls and Matlab engine function calls are blocking, which prohibits parallel execution of L1CTL protocol handling and baseband processing.

Instead, the interface *phyconnect* which we have developed for our physical layer development framework connects the Unix domain socket to Matlab via a memory mapped file, as depicted on the right hand side of Fig. 2. In order to prevent accidental overwriting of data in the memory, a handshake protocol has been implemented. Thus, phyconnect sending data to Matlab waits first for Matlab to retrieve any data in the memory mapped file. By the same token, Matlab waits for the phyconnect process to retrieve data first before overwriting it. The memory mapped file has a total length of 880 bytes, which is used to build an array of 220 entries of 32 bit unsigned integers. There are entries for all information that needs to be accessible by L1, *phyconnect* and L2, such as L1CTL message properties, payload, and necessary information for the handshake protocol. Thus, the proposed interface is a flexible solution to connect the L2/L3 layer of OsmocomBB with physical layer implementations in Matlab.

In the following section, we present our framework architecture that uses *phyconnect* to enable the simulation of our L1 realization in Matlab in combination with *mobile* (the application running L2/L3) of OsmocomBB.

5 Matlab Framework Architecture

The architecture of our baseband signal processing framework is shown in Fig. 2. It comprises a GSM physical layer implementation, referred to as *phydev*, and the interface *phyconnect*, which connects the *mobile* application of OsmocomBB, as explained in the previous section. Phydev is a Matlab realization of the physical baseband receiver that is typically implemented on the baseband processor, assisted by accelerator blocks in dedicated hardware (cf., Fig. 1). The main components of phydev are the L1 controller, the TPU and the three physical layer processing units *digital front-end* (DFE), *detector* (DET) and *decoder* (DEC). Note that this partitioning of the digital baseband signal processing into DFE, DET and DEC is also suitable for an integration in hardware, where controlling blocks that operate on the same input data type can be realized efficiently. DFE primitives operate on a stream of I-/Q-samples, which require no frame or burst alignment. DET functions, instead, require synchronized and buffered bursts of digital I-/Q-samples, typically at symbol rate $f_{sym} = 270.833$ Hz. The signal processing primitives of DEC operate on demodulated radio blocks of several hundred of bits. The bits are either represented by *hard-decisions*, i.e., logic '0' or '1', or by *soft-decisions* that represent the likelihood of a bit being rather a '0' or a '1'. Each physical layer processing unit consists of an individual controller and several primitive functions, that process specific signal processing tasks, such as channel estimation or channel decoding.

5.1 L1 Controller

The L1 controller builds the connection between the interface toward phyconnect and the main physical layer processing units DFE, DET and DEC. The L1 controller receives L1CTL messages from the socket of the interface via memory mapped file and creates confirmation (CONF) or indication (IND) messages for the layer above. The

[3] The basic messages are called *primitives of the physical layer* in the GSM specifications [8].

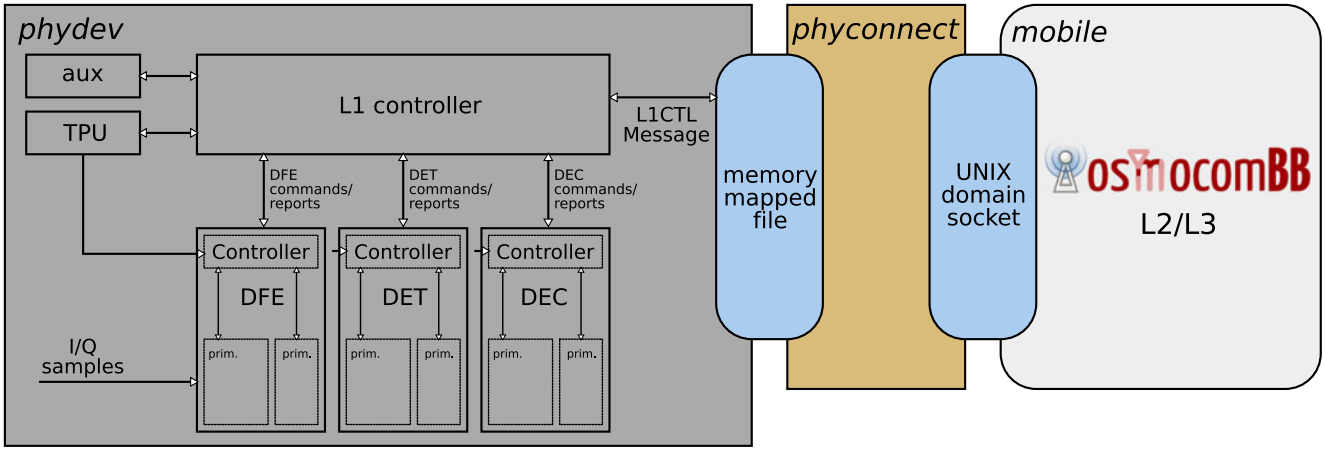


Figure 2 Architecture of the framework *phydev* and the interface *phyconnect* connected to OsmocomBB's *mobile* application via a memory mapped file and a Unix domain socket.

L1CTL messages are split up into *commands* for the corresponding DFE, DET and DEC controllers. These controllers parse the command and execute a sequence of signal processing primitives corresponding to an incoming command. The results of the primitives are collected and sent back to the L1 controller in a *report*. The L1 controller implements the PHY finite state machine of a MS for GSM, as specified in [8] and shown in Fig. 3.

After switching on the MS, the state machine starts in the NULL state. From this, after having received a PM_REQ message, the cell search procedure starts. First, the power levels of all possible GSM carriers are measured and reported to L2 in the *search BCH* state, in order to find the beacon carrier.

Next, synchronization in time and frequency is performed after having received a FBSB_REQ message. The controller sends corresponding commands to the units, where the primitive functions necessary for synchronization are executed, and evaluates the reports. In the *BCH* state the system information carried on the BCCH channel is extracted from the reports and sent to L2/L3. At this point the GSM state *camping on any cell* [10] is reached.[4]

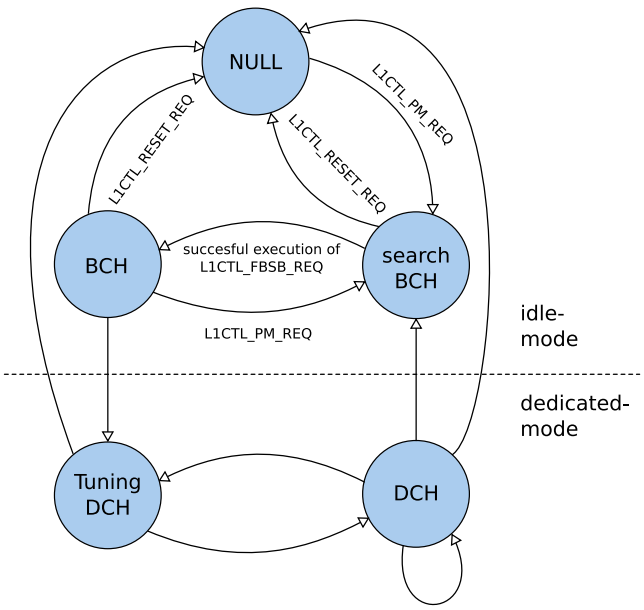


Figure 3 PHY finite state machine with the states for the idle mode and dedicated mode [8].

5.2 TPU

A TPU which keeps track of the four GSM timebase counters is necessary. Since several signal processing algorithms do not run fast enough in Matlab to be executed in real time, the TPU counters are emulated in software. This TPU allows the simulation of the timing between execution of signal processing blocks, controller procedures, and communication with higher layers. Each controller is able to read and update the TPU counter states. More specifically, according to the TPU counter state, messages are forwarded to the specific controllers of DFE, DET and DEC. The controllers in DFE, DET and DEC call the primitives according to the TPU counter states. At each call, the counters are incremented according to a processing time allocated to each primitive. This processing time is a measure for the actual time required for a typical hardware implementation (see Section 5.3.2).

[4]Note that the states of dedicated mode (see Fig. 3) are not implemented in our framework so far.

5.3 Digital Front-End (DFE)

Tasks of DFE include decimation filtering, DC-offset compensation, signal level measurements, carrier frequency synchronization for the cell-search procedure, and coarse synchronization in time. DFE is operating on a stream of non-synchronized I-/Q-samples. To ensure that the execution of operations on I-/Q-samples is carried out at the right time in this streaming based block, functions which provide a result after a pre-defined number of samples are necessary. The number of samples to be processed and other primitive-specific parameters are input arguments of the functions given from the DFE controller in terms of commands. The output of the primitive functions is forwarded to the DFE controller. The commands of the DFE controller and the primitives of DFE are explained in the following Sections 5.3.1 and 5.3.2 respectively.

5.3.1 Command Processing in the DFE Controller

The DFE controller receives the following commands from the L1 controller, calls the required primitives, and sends back a report.

- RXPWR (RX Link Level Measurement): MSs have to measure the received signal power on all GSM carriers served by the BTS, after power on as well as during operation. These measurements are performed by the *PM_meas()* primitive function. The DFE controller maps the result of the primitive to an integer value RX_LEV and computes the running average according to the GSM specifications in [11], which is reported to the L1 controller.
- FSYNCH_FB (Frequency Synchronization): carrier frequency synchronization is crucial during initial cell search and also during normal operation. The first step after power measurements is the decimation filtering on behalf of the *DEC_flt()* primitive. Subsequently the DC offset is estimated and removed with the *DC_comp()* primitive. The next step for the cell search procedure is the detection of the FB, transmitted on the beacon carrier. The finite state machine that corresponds to the frequency synchronization procedure is depicted in Fig. 4. FB detection is performed in the state *search FB* until either a FB is detected or a timeout is reached. Once a FB is found, the carrier frequency offset is estimated and the FB found flag and the estimated frequency offset are reported to the L1 controller.
- RX_NB/RX_SB (Normal burst / Synchronization burst): the first steps when processing NBs or SBs are the same as for the FSYNCH_FB message. In addition, the RX signal power is estimated with the PM_meas() primitive for the automatic gain control (AGC).

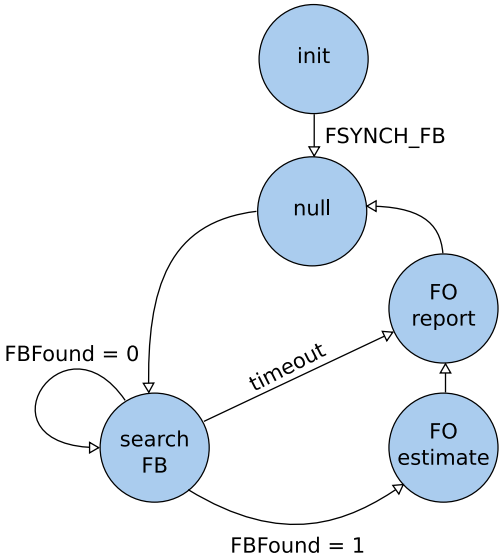


Figure 4 Finite state machine for processing a FSYNCH_FB message in the DFE controller.

5.3.2 Signal Processing Primitives in DFE

In this subsection, the signal processing primitives of the DFE will be explained.

- DEC_flt(): in our framework, we assume the input I-/Q-samples to arrive 2× or 4× oversampled from the analog to digital converter (ADC) and subsequent filter stages. In order to downsample the signal to symbol rate and avoid aliasing, decimation filtering is performed via Dec_flt(). The decimation filter of our framework is implemented as finite impulse response (FIR) filter with 32 taps (real-valued coefficients). The filter can be configured that decimation from 2× and 4× oversampling ratio is supported.
- DC_comp(): even state-of-the-art analog transceiver IC's for cellular communication do not provide a DC-free signal to the digital baseband. The residual DC offset can be dramatic and degrade detector performance significantly. The DC offset of the input signal is estimated by averaging over a given number of samples and subsequently subtracted.
- PM_meas(): the input arguments of PM_meas() are the number of samples to be processed and the frequency of the carrier to be measured. The output of PM_meas() is the RMS power in dBm computed over the amount of processed samples.
- FB_det(): the FB is transmitted as a complex sinusoid, which enables a variety of detection strategies at

receiver side. In this primitive, FB detection is performed according to [12], where the variance of the phase of a complex sinusoid is estimated and compared to a threshold value.

- FB_est(): the carrier frequency offset is estimated from the FB's complex sinusoid by using the T&F estimator from [13], which is based on angles of correlations. This approach has a significantly lower computational complexity when compared to costly periodogram-based frequency estimators. By calling the FB_est() primitive function, this carrier frequency offset estimation algorithm is executed by processing the number of samples specified as input arguments. The output of FB_est() is the estimated carrier frequency offset.

5.4 Detector (DET)

The DET unit is operating on synchronized and buffered bursts of I-/Q-samples at symbol rate. Its main task is the equalization and demodulation of NBs and SBs. More than that, the detection of the SB and the subsequent synchronization in time is performed in DET. In the following sections the command processing for the DET and its primitives are explained.

5.4.1 Command Processing in DET Controller

The DET controller receives the following commands from the L1 controller, calls and evaluates the corresponding primitives, and sends back a report.

- TSYNCH_SB (SB processing): after synchronization in frequency has been achieved, time synchronization needs to be performed. In GSM the detection of the SB which is also broadcast on the beacon carrier allows precise synchronization in time. A coarse timing estimate is already provided by the FB detection (see 5.3.2) and given as input argument to the *SB_synch()* primitive. Thus, the SB detection needs only to be performed on the part of the received samples, which has been identified by the coarse synchronization as potential SB. Once the exact location of the SB is known, its channel impulse response is estimated by use of the *ChEst()* primitive. Part of the GSM system information is transmitted on the SB and needs to be extracted during the cell search procedure. This payload is demodulated by means of the *Prefilter* and *Equalizer()* primitives and a report is sent back to the the L1 controller.
- RX_NB (Normal burst processing): similar to the TSYNCH_SB command, but without synchronization procedure, the training sequence of the processed NB is used to perform the channel estimation. Then, channel shortening, channel equalization and symbol demodulation are performed.

5.4.2 Signal Processing Primitives in DET

- ChEst(): in our implementation the channel profile is estimated in a least-squares sense, as described in [14], with the aid of the training sequence of each GSM burst.
- Prefilter(): the channel impulse response of a received GSM signal might last over several symbol periods due to inter symbol interference. In order to enable reduced state sequence estimation (RSSE) in the equalizer, the channel energy must be concentrated within the first few channel taps. To this end, a channel shortening prefilter according to [15] is applied before equalization.
- Equalizer(): in the equalizer we employ trellis based equalization and symbol demodulation. Since the number of trellis states in an equalizer based on maximum likelihood sequence estimation (MLSE) [16] is prohibitively large, especially for non-GMSK modulations, RSSE according to [17] is used. A straight-forward approach would process one single trellis with 148 symbols. This approach has the drawback that the known training sequence is processed although it is already known at the receiver. We have implemented a reduced complexity approach, where two sub-trellises with 58 symbols each are processed independently by using the training symbols as additional tail bits.
- SB_synch(): the position of the 64 bit extended training sequence of the SB is detected accurately by performing a correlation between the received signal and the known training sequence. The output of this primitive is the index with the highest correlation peak. It is assumed that this index indicates the start of the SB what allows precise burst and frame alignment.

5.5 Decoder (DEC)

The DEC signal processing unit performs functions such as deinterleaving, bit swapping, demapping or channel decoding with a Viterbi decoder. DEC receives bursts from DET which consist of hard- or soft-decisions. Internally, DEC processes radio blocks or SBs.

A framework of controller and primitives inside DEC has been setup with the prospect of GSM extensions such as EDGE or even Evolved EDGE. As a matter of proof, the receiver in this work was built to be able to achieve a camping on any cell state. Naturally, the DEC framework is exploited only partially but can be enhanced with full GSM functionality or even to the above mentioned GSM extensions.

5.5.1 Command Processing in DEC Controller

There exist two different commands for DEC:

- SB_DEC (Synchronization burst processing): this is the command for the decoding of the SB. Only the *ChanDec()* primitive is used.
- NB_DEC (Normal burst processing): this command is used for the decoding of traffic and control channel data. As a radio block consists of four bursts, a NB_DEC command is required four times in succession. Consequently, as long as a radio block is not yet complete, the DEC controller calls only the *Demult()* primitive. However, as soon as a radio block is complete, the *DeswapDemap()*, *Deint()*, *Depunc()*, *IR()*, and *ChanDec()* primitives are required, as well.

5.5.2 Signal Processing Primitives in DEC

- Demult(): the Demult() primitive collects equalized and demodulated bursts until a radio block is complete and can be further processed. In addition, it organizes the demultiplexing of various bursts onto the correct radio block for voice or multislot data traffic channels. However, the latter functionality has not yet been implemented in this work.
- DemapDeswap(): this primitive extracts (demaps) a portion of hard- or soft-decisions from a radio block. This is necessary as not all hard- or soft-decisions within a radio block can or need to be processed at once. EDGE [5] radio blocks, for example, require that the header portion of a radio block be processed first. Only then, the remaining hard- or soft-decisions can be further processed. However, only the case of SB and BCCH carrier have been implemented in this work. The demapping primitive can be combined with a swapping operation. The latter consists of swapping bit positions within a radio block or withing a burst with the help of lookup tables. However, SB and BCCH do not require bit swapping operations. Correspondingly, this operation was not further pursued in this work.
- Deint(): deinterleaver functionality is provided by this primitive. The GSM specifications dictate interleaver operations usually by a formula, sometimes by means of a lookup table.
- Depunc(): radio blocks or parts thereof which were punctured at the transmitter need to be depunctured at the receiver. In case of soft-decisions, the puncture inverse consists of zero insertions at the corresponding bit positions. Nevertheless, BCCH radio blocks are not punctured.
- IR(): IR operations, the combination of retransmitted data with previously stored data, are performed by this primitive. But, IR is not defined for the SB and BCCH. Therefore, no further work was put into this primitive.
- ChanDec(): the data in GSM is encoded with a half rate convolutional code. Parity bits are added for error detection. In ChanDec(), a Viterbi decoder decodes the radio block. Subsequently, it performs a parity check.

5.6 Auxiliary Functions

In addition to the primitive functions, auxiliary functions (labeled *aux* in Fig. 2) for common transceiver operations like RF power control or oscillator tuning (*DCXO_tune*) are provided. In Matlab where no physical DCXO is present, the frequency tuning was performed by rotation via a complex exponential.

5.7 Examples of L1CTL Message Processing in Phydev

In this section, the processing of a FBSB_REQ and the reception of a BCCH radio block are explained. The latter leads to a DATA_IND message.

5.7.1 FBSB_REQ

The processing of this message is depicted in Fig. 5.

After receiving a FBSB_REQ message, the L1 controller sends a FSYNCH_FB command to DFE. DFE processes the FSYNCH_FB command and creates a corresponding DFE_rpt. After the L1 controller receives the DFE_rpt, it sends a RX_SB command to DFE, which executes the RX_SB command and creates a corresponding DFE_rpt (not shown in Fig. 5). The RX_SB chain with respect to DFE is identical with the DFE part of Fig. 6 by replacing the RX_NB with a RX_SB command. After the L1 controller receives the corresponding DFE_rpt, a TSYNCH_SB command is sent to DET. Again, as soon as the L1 controller receives the DET_rpt, it configures DEC with a SB_DEC command. Upon reception of the DEC_rpt the L1 controller generates a FBSB_CONF message for higher layers.

5.7.2 DATA_IND

After synchronization in frequency and time, the L1 controller knows according to its counter states and the GSM specifications, when to receive the bursts pertaining to a BCCH radio block. The steps until a DATA_IND message is created are depicted in Fig. 6.

Naturally, this behavior is only possible in the *BCH* state (see Fig. 3). When the BN and FN counters reach the time slot of the first burst of a BCCH radio block, the L1 controller sends a RX_NB command to DFE. Subsequently, is sends a RX_NB command to DET. Lastly, DEC receives

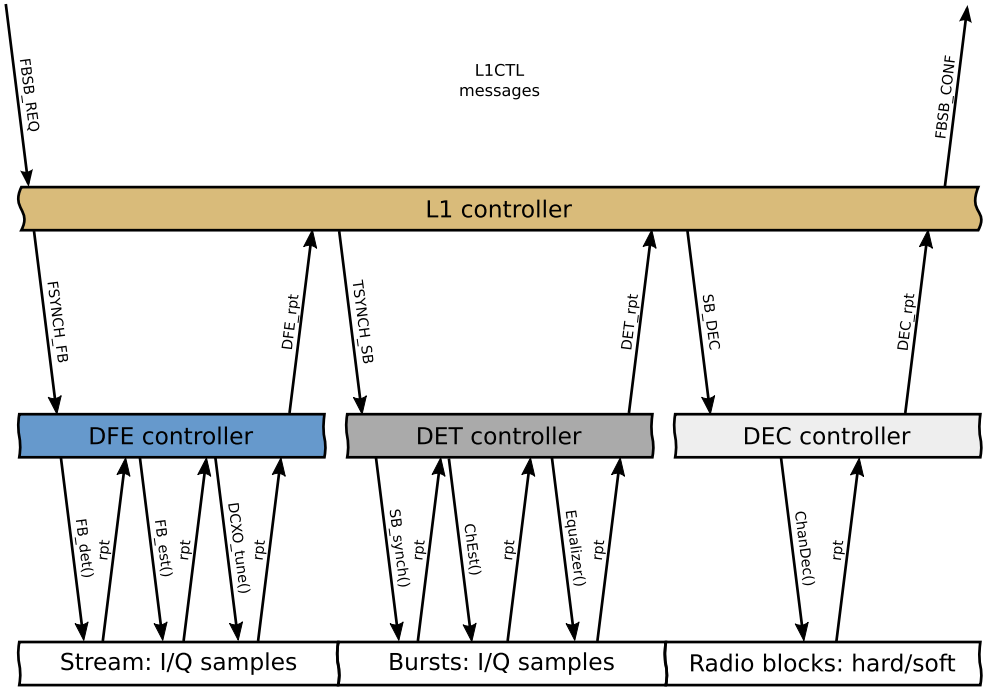


Figure 5 Processing of FBSB_REQ message. The RX_SB processing with respect to DFE is omitted. It is identical to the DFE processing in Fig. 6, just that RX_NB needs to be replaced with a RX_SB command.

a NB_DEC command. Next, when the BN and FN counters reach the time slot of the second burst of the BCCH radio block, the commands for DFE, DET, and DEC are repeated. This needs to be repeated two more times until the radio block is complete. Only now, DEC can process the radio block. Upon reception of the DEC_rpt the L1 controller generates a DATA_CONF message for higher layers.

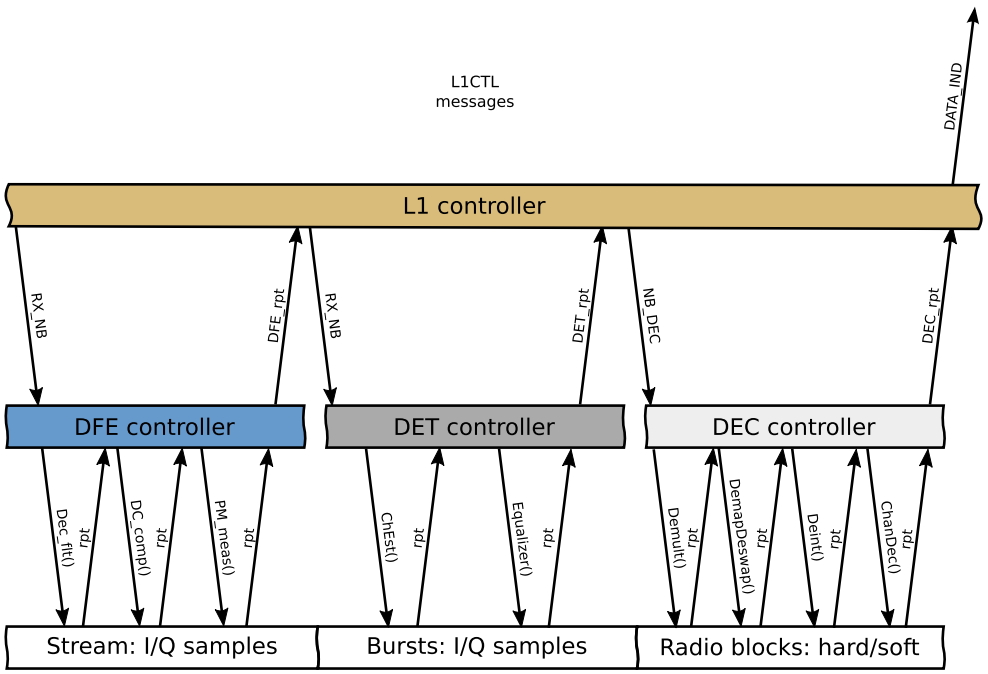


Figure 6 The reception of a BCCH radio block which leads to a DATA_IND message: the commands for DFE, DET, and DEC must be sent several times until a radio block is complete. Only then, DEC can start further processing.

6 *Phydev* and OsmocomBB Embedded

When running *mobile* and *phydev* on a PC, it is not possible to process received samples from a BTS in real-time, since the execution of the signal processing primitives in Matlab is not fast enough. Therefore, we have developed an embedded system that allows real-time execution, where *mobile*, *phyconnect* and *phydev* are implemented on FPGA and PowerPC.

For the hardware implementation of phydev, we have used a Xilinx ML605 development board that comprises a Virtex6 FPGA.[5] The ML605 is attached to a transceiver board with a state-of-the-art multi-band RF transceiver,[6] in order to capture RF signals transmitted from a BTS over the air, and convert them to baseband signals. The baseband signals are processed by our *phydev* realization on Virtex6 FPGA, which is connected to a PowerPC core, where *phyconnect* and *mobile* are running on an embedded Linux (cf., Fig. 7).

To this end, the *mobile* application was ported to the embedded Linux by cross-compiling the software for PowerPC. This cross-compilation requires a reconfiguration of the GNU/Autotools, that are used by OsmocomBB for the building process. As the embedded Linux does not support shared libraries, only static libraries were used in the building process. By assigning a new host argument to the *configure* script and adding the *disable-shared* option, tailored Makefiles for the PowerPC are generated.

In addition, *phyconnect* and a L1CTL test program (as shown in Fig. 7) were built for the embedded Linux. With the L1CTL test program, L1CTL messages can be sent via *phyconnect* to the L1 controller running on the embedded *phydev*. The L1 controller on the FPGA receives messages from and sends messages to the embedded version of *phyconnect*. Similar to the software implementation where a memory mapped file is used for this message exchange, a register bank implemented on the FPGA is mapped to the address space of the CPU. Some of the registers are reserved for the handshake protocol, resembling the communication between Matlab and phyconnect over the memory mapped file (cf. Section 4).

6.1 TPU Embedded

As previously explained, the L1 controller needs to have access to GSM counters of the TPU. In our embedded hardware implementation, the GSM counters are divided into two parts. The QN counter is directly coupled to the 26 MHz clock of the RF transceiver by counting 24 clock cycles per quarter symbol (TPU 1). The second part (TPU 2) consists of BN, TN and FN counters and is directly integrated into the finite-state machine of the L1 controller. These counters are incremented based on timing events which are generated whenever the QN counter is restarted after having reached the maximal value of 624 at the end of a timeslot.

Clearly, real-time processing requires real-time acquisition of baseband data. The RF receiver must be switched on and off at the correct time instance to have the desired part of the sequence of signal samples available in the baseband. To this end, all interactions with the RF transceiver are stored in an *event table* along with a time stamp. An event is triggered as soon as the QN counter reaches the value of the corresponding time stamp. Different event tables are stored in TPU 1 and each table is dedicated to a specific scenario, such as receiving a NB, transmitting a NB or synchronization. Control registers in TPU 1 contain specific configuration values, that are accessed when certain events are triggered. Thus, the required commands for the RF transceiver IC are generated when a certain event is triggered by reading the corresponding control register at the moment of execution. With this approach, the L1 controller can configure the commands generated by certain events by simply updating the content of the control register within TPU 1. The *reset QN* event, for example, evaluates

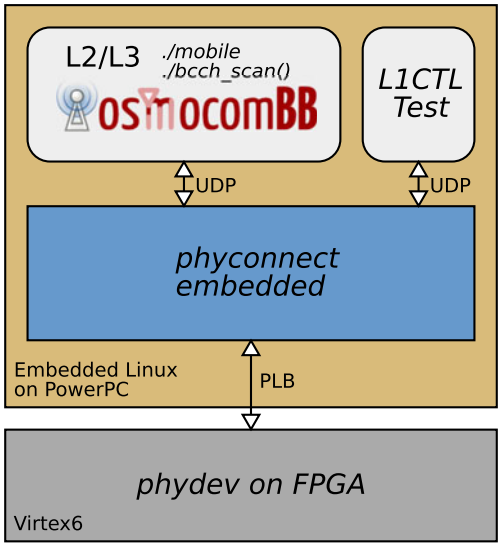


Figure 7 Embedded OsmocomBB, phydev and phyconnect. Phydev is connected to phyconnect via the PLB and phyconnect to OsmocomBB and the L1CTL Test application via UDP sockets.

[5]Note that we have realized this embedded version of our system on FPGA, because we are aiming at a baseband ASIC in the near future.

[6]IRIS305 RF Transceiver from Advanced Circuit Pursuit (ACP) AG, Zollikon, Switzerland.

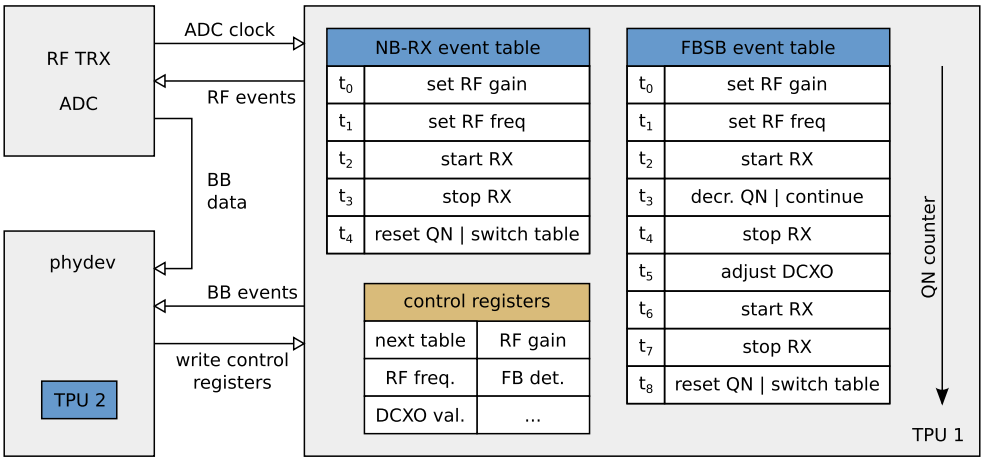


Figure 8 The embedded TPU consists of two parts: a cycle-true QN counter (TPU 1) and an event-based TPU 2 for BN, TN, and FN counters.

the content of the *next table* register before resetting the QN counter. Thus, the L1 controller can bring the RF transceiver to a different mode of operation by selecting a different event table.

The example in Fig. 8 shows an event table for synchronization in frequency and time (FBSB), as well as an event table required for the reception of a NB. At the beginning of the FB detection, the desired gain and frequency of the RF transceiver is configured according to the corresponding control registers. When the QN counter reaches time stamp t_2, the ADC is turned on and baseband data is directly passed to phydev. At t_3, the control register *FB det* is evaluated. The QN counter is decremented to $t_3 - \delta$ such that the ADC stays turned on and as soon as *FB_det()* has detected

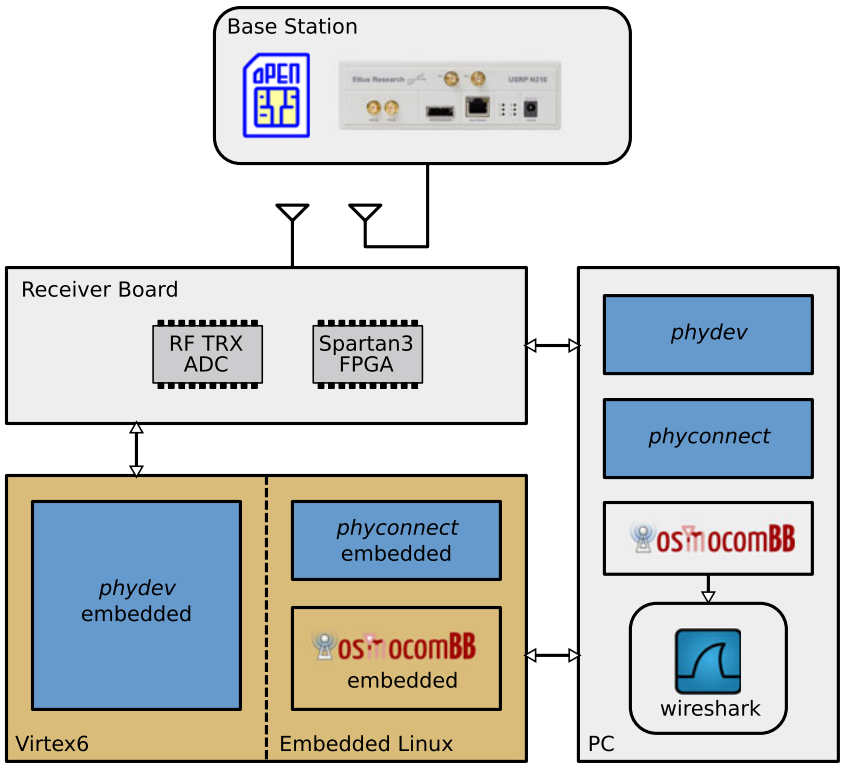


Figure 9 Testbed architecture with involved components. Phydev, phyconnect and OsmocomBB on Virtex6 and the embedded Linux enable real-time operation. Corresponding implementation on the PC allows offline operation.

a FB, the control register is updated and the TPU is commanded to continue to t_4 where the ADC is turned off. The DCXO of the RF transceiver is corrected according to the result of *FB_est()* before the ADC is turned on again to sample the SB. Before t_8 is reached, the L1 controller must have decided whether another synchronization run is required. Otherwise, the value of the *next table* register is updated and the TPU starts to operate in synchronized mode and event tables for receiving or transmitting NBs are employed (e.g. the NB-RX event table depicted in Fig. 8). The *reset QN* command at the end of each table triggers an event towards phydev which is used by TPU 2 to update BN, TN, and FN counters.

6.2 Testbed Setup

The embedded version of our system has been integrated in the testbed setup illustrated in Fig. 9. The testbed setup allows the verification of our baseband signal processing framework with real-world data.

The testbed contains a GSM BTS emulator, that has been realized with the open source software OpenBTS [18] and *GNU radio* running on a PC, and a USRP[7] board with antenna to transmit the signal over the air. The testbed allows the operation in real-time with embedded phydev, and phyconnect, as well as offline operation by running L1/L2/L3 on a PC.

The baseband I/Q samples coming from the receiver board are processed in real-time by our embedded system. In offline mode, *phydev*, *phyconnect* and OsmocomBB run on a PC and process samples recorded with the receiver board, or samples generated with a GSM transmitter software implementation in Matlab.

In both modes, the *mobile* application encapsulates the down-link data in UDP packets via GSMTAP,[8] which are forwarded to the Wireshark [19] protocol analyzer running on the PC for visualization. A picture of the testbed setup is shown in Fig. 10.

The testbed setup has been used to verify the functionality of our framework in combination with L2/L3 of OsmocomBB by performing the initial cell search procedure in GSM (cf. Section 5.1). To this end, the BTS emulator transmits a standard-compliant GSM beacon carrier. The signal is received and processed on the receiver board, as previously described, and corresponding I/Q samples are loaded from the receiver board into *phydev*, where synchronization in time and frequency is achieved. The GSM system information broadcast on the beacon carrier is correctly extracted and propagated through OsmocomBB to Wireshark, where the SI messages are displayed.

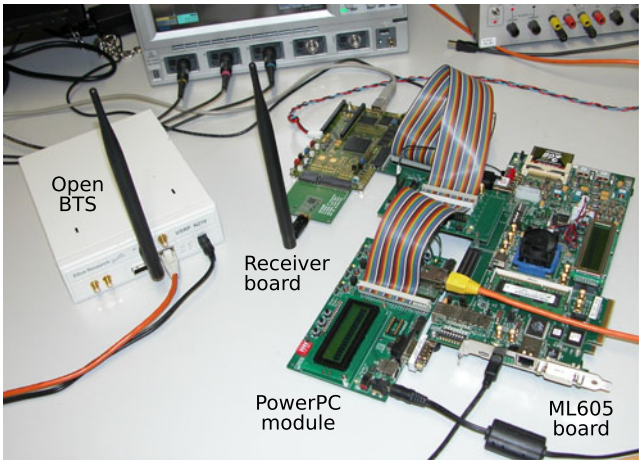


Figure 10 Setup of the testbed with USRP N200 BTS, PowerPC module, ML605 FPGA board and receiver board.

7 Conclusion

With the proposed baseband signal processing framework we offer a versatile development and exploration environment targeting baseband implementations for DSPs, FPGAs or ASICs. It is explained why it is favorable to develop baseband algorithms and architectures within a complete system enabling access to higher protocol layers. The framework shows how the physical layer can be realized in Matlab in order to obtain a viable environment for baseband algorithm development prior the deployment on a hardware platform. It is further illustrated how it can be connected to OsmocomBB with a dedicated interface. In addition to the offline version, we developed an embedded version of the framework on FPGA and PowerPC, which is part of a testbed that allows real-time operation with GSM signals transmitted by an OpenBTS base-station emulator over the air.

Acknowledgments We would like to thank Dominic Just and Pirmin Vogel for their valuable work during their student projects and Raphael Rolny for this consultation regarding user cooperation. We thank ACP AG for providing us the IRIS305 single-chip RF transceiver for our testbed setup. In addition, we want to thank David Tschopp and

[7] Universal Software Radio Peripheral, from Ettus Research.

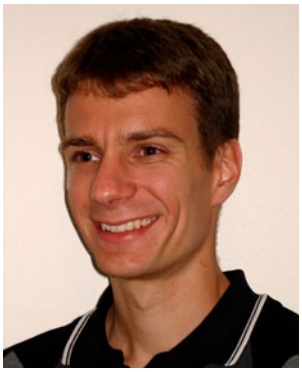
[8] A pseudo-header used to transport GSM frames of the Um air interface over UDP/IP.

Dominik Riha for their support on the receiver board. This work was funded by CTI, Switzerland, in collaboration with ACP AG.

References

1. Kröll, H., Benkeser, C., Zwicky, S., Weber, B., Huang, Q. (2012). Baseband signal processing framework for the OsmocomBB GSM Protocol Stack. In *Wireless innovation forum European conference on communication technologies and software defined radio*. Brussles, Belgium.
2. OsmocomBB (2012). An Open Source GSM Baseband software implementation. http://bb.osmocombb.org.
3. Chang, L.F., & Wang, Y. (2009). EDGE incremental redundancy memory structure and memory management. US Patent App (Vol. 12, 507, p 835).
4. 3GPP TR 44.060 (2009). General packet radio service (GPRS); mobile station (MS) - base station system (BSS) interface; radio link control / medium access control (RLC/MAC) protocol. December.
5. Seurre, E., Savelli, P., Pietri, P.J. (2003). *EDGE for mobile internet*. Norwood: Artech House Publishers.
6. Djeumou, B., Lasaulce, S., Klein, A.G. (2007). Practical quantize-and-forward schemes for the frequency division relay channel. *EURASIP Journal on Wireless Communications and Networking*, *2007*, 2.
7. 3GPP TR 45.001. GSM/EDGE radio access network; physical layer on the radio path; general description, November 2009.
8. GSM/EDGE layer 1; general requirements, December 2009.
9. ISO/IEC 13239. Information technology telecommunications and information exchange between systems high-level data link control (HDLC) procedures, July 2002.
10. 3GPP TR 43.022. Functions related to mobile station (MS) in idle mode and group receive mode, December 2009.
11. 3GPP TR 45.008. GSM/EDGE radio access network; radio subsystem link control, November 2009.
12. Kröll, H., Zwicky, S., Benkeser, C., Huang, Q., Burg, A. (2012). Low-complexity frequency synchronization for GSM systems: Algorithms ad implementation. In *IV international congress on ultra modern telecommunications and control systems 2012 (ICUMT 2012)* (pp. 175–180). St. Petersburg, Russia.
13. Tufts, D.W., & Fiore, P.D. (1996). Simple, effective estimation of frequency based on Prony's method. In *Proceedings of IEEE international conference on acoustics, speech, and signal processing (ICASSP)* (Vol. 5, pp. 2801–2804).
14. Yakhnich, E. (2001). Channel estimation for EGPRS modems. In *Vehicular technology conference, 2001. VTC 2001 spring. IEEE VTS 53rd* (Vol. 1, pp. 419–422). IEEE.
15. Gerstacker, W.H., Obernosterer, F., Meyer, R., Huber, J.B. (2000). An efficient method for prefilter computation for reduced-state equalization. In *Personal, indoor and mobile radio communications, 2000. PIMRC 2000. The 11th IEEE international symposium on* (Vol. 1, pp. 604–609). IEEE.
16. Proakis, J.G. (1987). *Digital communications*. McGraw-hill.
17. Eyuboglu, M.V., & Qureshi, S.U.H. (1988). Reduced-state sequence estimation with set partitioning and decision feedback. *IEEE Transactions on Communications*, *36*(1), 13–20.
18. OpenBTS. http://openbts.sourceforge.net, cited July 2012.
19. Orebaugh, A., Ramirez, G., Burke, J. (2007). *Wireshark & ethereal network protocol analyzer toolkit*. Syngress Media Inc.

Harald Kröll received his Dipl.-Ing. degree (summa cum laude) in Electrical Engineering from Graz University of Technology in 2010. During his Master's thesis he worked on Ultra Wideband Location Fingerprinting at the Wireless Communications Group at ETH Zurich. In the same year he joined the Integrated Systems Laboratory (IIS) at ETH Zurich where he is working towards a PhD degree. His research interests include signal processing and physical layer architectures for wireless communication systems such as Evolved EDGE and LTE-Advanced.



Stefan Zwicky received the MSc degree in Information Technology and Electrical Engineering in 2007 from the Swiss Federal Institute of Technology, Zurich. He then joined Celestrius AG, an ETH-spinoff in the field of MIMO wireless communication, where he worked in the digital ASIC development. He is currently a PhD student in the Integrated Systems Laboratory (IIS) at ETH Zurich. His research interests include signal processing for mobile communication systems and the design of VLSI circuits and systems.

Benjamin Weber received his BSc and MSc in Electrical Engineering and Information Technology from the Swiss Federal Institute of Technology (ETH) in August 2010 and in June 2012, respectively. Currently, he is a PhD student at the Department of Information Technology and Electrical Engineering at ETH, more particularly, at the Integrated Systems Laboratory. His research interests include low-power physical layer architectures, open-source protocol stacks, and cross-layer optimization in cellular communications.

Christian Benkeser was born in Bhl/Baden, Germany, in 1977. He received his Dipl.-Ing. degree in electrical engineering from the Karlsruhe Institute of Technology (KIT), Germany, in 2004. In the same year, he joined the Integrated Systems Laboratory (IIS) of the Swiss Federal Institute of Technology (ETH) Zurich, Switzerland, from where he graduated with the Dr. sc. degree in 2009.

From 2004 to 2009, he was a research assistant with the IIS, and a consultant for Advanced Circuit Pursuit (ACP) AG, an IC company in RF transceivers for cellular communications. From 2009 to 2012 he held positions as postdoctoral researcher at IIS and as senior design engineer at ACP AG. During this time he was leading a group of system and design engineers developing VLSI circuits and systems for wireless communications. In 2013, he joined RUAG Space, an independent supplier of space technology, where he is currently working as system engineer for opto-electronical systems for space applications. His research interests include signal processing, circuits and systems for wireless communication and space applications.

Qiuting Huang received his Ph.D. degree in applied sciences from the Katholieke Universiteit Leuven, Belgium, in 1987. Between 1987 and 1992 he was a lecturer at the University of East Anglia, Norwich, UK. Since January 1993, he has been with the Integrated Systems Laboratory, Swiss Federal Institute of Technology (ETH), Zurich, where he is Professor of Electronics. In 2007 he was also appointed as a part-time Cheung Kong Seminar Professor by the Chinese Ministry of Education and the Cheung Kong Foundation and has been affiliated with the South East University, Nanjing, China.

Prof. Huang's research interests span RF, analog, mixed analog-digital as well as digital application specific integrated circuits and systems, with an emphasis on wireless communications applications in recent years. He has published widely on those topics in leading solid-state circuits conferences and journals. He is a member of the technical program committees of the International Solid-State Circuits Conference (ISSCC) and the European Solid-State Circuits Conference (ESSCIRC). He is also a member of the executive committee of ISSCC.