

# Implementing 3G Layer 1 Signal Processing (Tx)

Tim Howe

Plextek Ltd, London Road, Great Chesterford, Essex, CB10 1NY  
tdh@plextek.co.uk

*This paper begins by reviewing the signal processing functions within Layer 1 of the evolving 3G wireless standard. It makes a qualitative assessment of the algorithms involved and outlines some of the issues encountered by Plextek during their software implementation of the Tx path for FDD. A prototype CPU / FPGA hybrid solution is then presented. Note: This paper does not consider Layer 1 from the perspective of the handset manufacturer, nor does it cover TDD.*

## ABBREVIATIONS

CCTrCH	-	Coded composite Transport Channel
CRC	-	Cyclic Redundancy Check
DL	-	Downlink
DTX	-	Don't Transmit qualifier
FBI	-	Feedback information
FEC	-	Forward Error Correction
TPC	-	Transmit power control
TTI	-	Transmission Time Interval
UL	-	Uplink

## OVERVIEW OF LAYER 1 PROCESSES

Figure 1 below shows the key functions that comprise Layer 1. This paper focuses on the Tx process chain but much of what is said also applies in the Rx direction, although the code synchroniser, RAKE and FEC decoders add significantly to the Rx processing requirement.

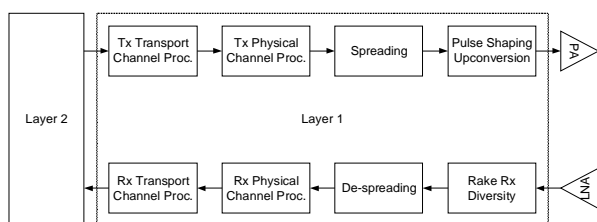


Figure 1: 3G Layer 1 Data Path

## Uplink and Downlink

There are major differences between uplink and downlink, both in terms of the air interface and how the channels are processed. Base stations and handsets need only take care of one direction. Test equipment on the

other hand may have to deal with both in which case some commonality may be exploited.

## Transport Channel Processing

Data is passed between Layer 2 and the transport channel processor in the form of transport block sets. Each active transport channel will pass a set in each direction at a rate defined by its TTI, which may be 10, 20, 40 or 80ms.

Data is in the form of bits or bit-arrays throughout. Implementation efficiency will therefore depend on effective storage and manipulation of bit-oriented data.

There is a good degree of commonality between UL and DL at algorithm level but the process ordering and scheduling is different. This may be dealt with as two separate implementations or a 'dual mode' implementation of increased complexity.

## CRC

A CRC of length 0, 8, 16 or 24 bits is added to each transport block on Tx and evaluated to check data integrity in the Rx. The CRC algorithm may be viewed as a bit-wise logical operation with feedback and is the same for UL and DL.

## Channel Coding

The transport blocks with their CRC's are merged then split into code blocks. Each code block then passes through a block based FEC algorithm with a choice of 1/2 rate convolutional, 1/3 rate convolutional, 1/3 rate Turbo or no coding. The FEC algorithms are the same for UL and DL.

## Rate Matching

The purpose of rate matching is to adapt the amount of data emerging from the FEC block to fit the amount of physical channel capacity that has been made available

for the CCTrCH (user / connection). The operation of UL and DL differ significantly here.

Puncturing or repetition is used to adapt the amount of data in a frame (UL) or TTI (DL). The puncturing / repetition itself is mechanical and easy to implement. The rate matching calculations that control it are, however, quite complicated and best dealt with in software. Dynamic changes in channel configuration during the lifetime of a connection adds further complication and requires of all the rate matching parameters to be recalculated.

After rate matching in the DL, a second stream termed 'DTX' will now accompany the data stream for each channel. Furthermore, in the case of compressed mode by puncturing there will be a third stream of 'p' bits but these are almost immediately stripped out again in the 1<sup>st</sup> Interleaver.

### 1<sup>st</sup> Interleaver

Simple block interleaving is then applied before the data is broken into frames ready for mapping onto the physical channels. The 1<sup>st</sup> Interleaver therefore forms the interface between TTI rate and frame rate processing in both UL and DL and is functionally similar for each.

1 TTI worth of data storage is required per channel at this point.

### CCTrCH MUX

This simple process brings together a group of transport channels that are to be mapped onto the set of physical channels associated with a given user / connection (only one CCTrCH is allowed in the UL).

### Physical Channel Mapping

Each frame of data emerging from the CCTrCH is fragmented across its associated set of physical channels. The rate matching performed earlier ensures that this data fits exactly.

### 2<sup>nd</sup> Interleaver

This interleaver disperses the data across the 15 slots of the frame.

### Spreading and Channel Gain

These processes are totally different in the UL and DL.

In the uplink, each physical channel is essentially a BPSK channel which may be mapped onto the I or Q modulation axis and has its own relative gain and channelisation code.

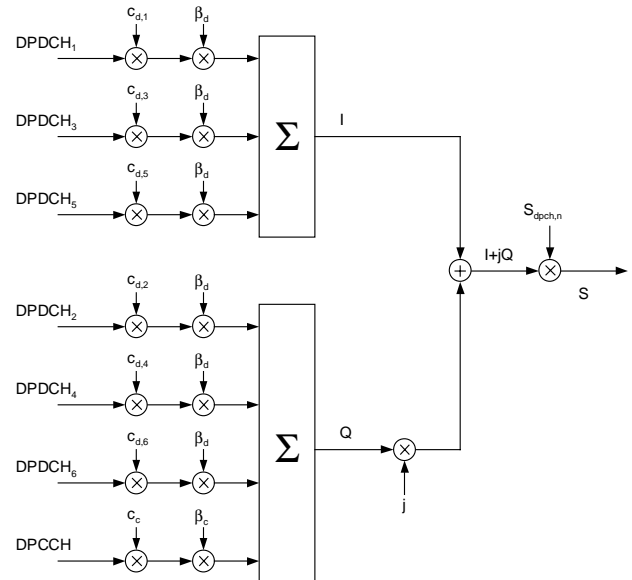


Figure 2: Channelisation, Gain and Spreading for Uplink Physical Channels

Scrambling is then performed on the complex sum and the final gain (including inner loop power control) is applied.

In the downlink, each physical channel is fed into the following block resulting in QPSK modulation. The control channel is time division multiplexed in with the data channel (slot format dependent).

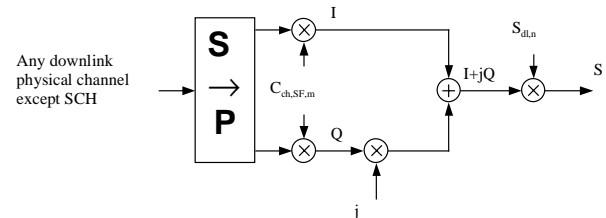


Figure 3: Channelisation and Spreading for Downlink Physical Channels

Multiple DL physical channels are summed after scrambling and have their gains applied (see Figure 4). It is possible for the relative gain, compressed mode gain and inner loop power control gain to all be applied at this point.

Note: Physical channels belonging to the same CCTrCH (Coded Composite Transport Channel) may have the gain for their control channel bits set to zero on all but the first physical channel (termed 'multicode transmission').

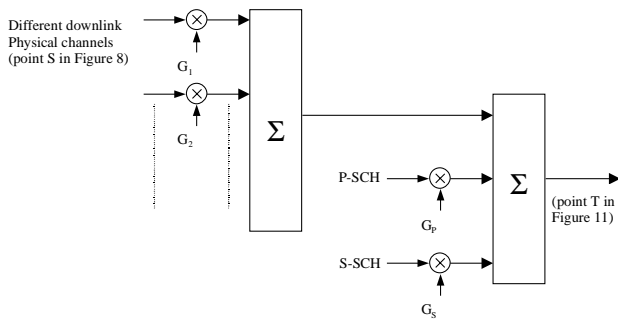


Figure 4: Summing and Gain for Downlink Physical Channels

The channelisation and scrambling codes themselves are generated by means of bit-oriented shift register style algorithms.

### IMPLEMENTATION IN SOFTWARE

Plextek Ltd has completed an all-software implementation of the Tx process chain that provides UL and DL functionality for a 3G test equipment. Here the emphasis was on flexibility and ease of modification to track future changes in the standards.

### Target Platform

The platform used was a cluster of 4 Texas Instruments TMS320-C6202 fixed point DSP devices, defined by the client and arranged as shown in Figure 5. The inter-processor paths were provided by an FPGA and an additional 8MB of SDRAM was available to each DSP.

Division of the cluster into 1 transport and 3 physical channel processors was done at an early stage in the system design based on the anticipated computational requirement for transport and physical channel processing.

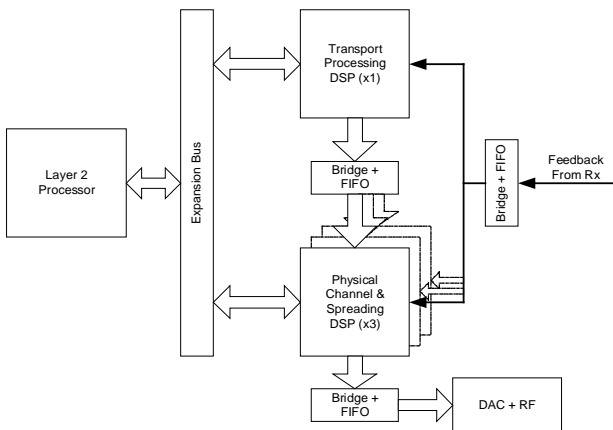


Figure 5: Topology of DSP Platform

### Software Architecture

Layer 1 is a real time system composed of a number of processes all operating in parallel. Software implementations of such systems in general require a scheduler to drive process execution in an order that meets the system's real time constraints. For this purpose, the TI DSP BIOS was used to provide simple pre-emptive, priority based scheduling.

The priority of thread used to run each process was chosen to be inversely proportional to the period in which its execution must be completed. This gives rise to an order of process execution that dynamically adapts to suit the system configuration and load (auto load balancing).

### The Unseen Code

On casual observation, what is most definitely not clear from the 3G specifications is the amount of control infrastructure that must surround the core data processes in order to create a working system. (This was perhaps more true for our implementation that had to provide full functionality for both UL and DL.) Functions such as memory management, error handling and interface control must also be provided.

In our implementation, the volume of control code exceeded that of the data processes by a factor of 2.

### Code Efficiency

The multi-mode nature of most of the data processes resulted in yet more control-style code. Optimised for DSP, both the TI C compiler and the processor core itself perform badly when dealing with this type of code. This resulted in significantly lower CPU performance and code density to that expected (and actually achieved) for the more algorithmic data processes.

The compiler coped well with 'clean' data processing loops. Loops that contained decisions needed to be rewritten with the decisions outside. Certain operations, such as those that do not map cleanly onto groups of  $2^n$  bits in a word (such as the 3 into 1 stream interleave at the output of the turbo encoder) were very difficult to optimise, even by hand.

### Efficiency of Bit-Oriented Operations

Due to the need to make very conservative use of the limited on-chip data RAM, data bits had to be packed into 32 bit words.

The TI C6202 core has a severe lack of bit-processing capability. This combined with the packing of data into words made most of the bit-oriented algorithms messy to implement. Good efficiency could only be obtained in

places where bits could be processed in blocks (whole words). This excluded many of the algorithms with feedback dependency.

### Chip Rate Processing

The need to speed operations as much as possible for the S/W implementation involved merging the channelisation and scrambling sequences with an XOR operation then applying all but the inner loop power control gain in one go, avoiding the need for multiplies. The state of DTX causes either 0 or  $G_r$  to pass into a complex sign changer which in turn yields 0 or  $\pm G_r + 0$  or  $\pm j.G_r$ .

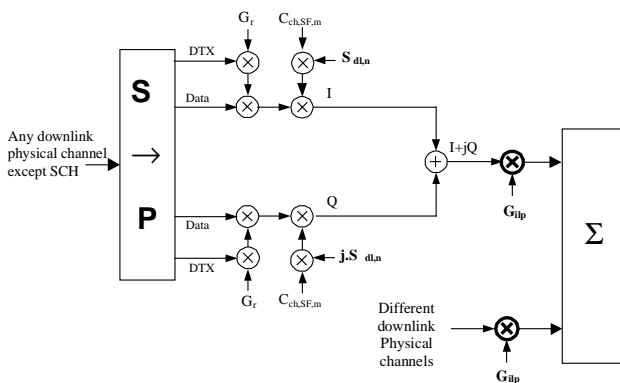


Figure 6: Re-arrangement of Scrambling and Gain Operations (DL)

The Inner Loop Power Control gain then gets applied prior to the final channel summation by means of a full complex multiply (as  $G_{ip}$  is itself complex in the DL).

The chip-rate spreading and gain processes mapped well onto the DSP architecture, as they were word as opposed to bit-oriented. Assembly language was required in places to achieve performance goals.

### Strengths of the S/W Implementation

The traditional benefits of a software implementation still applied: Ease of code maintenance, accessibility of large amounts of data storage (off-chip RAM) and the ability to deal with complicated algorithms and/or floating point.

The pre-emptive scheduler provides an excellent solution to the problem of dynamically managing multiple logical channels and processes with different rates of execution.

### FPGA BASED IMPLEMENTATION

Figure 7 shows an experimental hybrid CPU / FPGA solution consisting of a PowerPC based control processor and two FPGAs, one dealing with channel coding and multiplexing and one with channelisation, spreading and gain.

### FPGA Transport Processor

The transport processing is performed in one FPGA that is coupled to an external RAM to provide the necessary data storage. Arbitration over RAM access and management of bit-arrays is handled by the 'Buffer and I/O Management' block.

The FPGA deals effortlessly with the bit-oriented data processing. Processing data for multiple channels with potentially different configurations and rates is more difficult as it is not possible for the data process chains to 'context-switch'. Instead, sequential process scheduling is achieved by means of control tables (containing pointers and parameters) that are fed by the CPU. There are 4 parallel processing chains, nominally reserved for 80, 40, 20 and 10ms TTI intervals. A load-balancing algorithm in the CPU dynamically overrides this mapping to improve efficiency.

In contrast to the software implementation of the Turbo code interleaver, the FPGA version follows the 3G specifications more directly and uses parameters calculated by the CPU.

The data storage requirement of the transport channel processor is dependent on many factors including the number of active channels, their data and FEC rate and their TTI interval. A 2Mbps channel with 80ms TTI and rate 1/3 Turbo coding requires around 800kbits storage.

### Channelisation, Gain and Scrambling FPGA

The second FPGA is fed with Dedicated Physical CHannel data slot at a time and performs channelisation, spreading, application of relative gain and channel summing. The output consists of complex chips (I and Q) ready for root-raised cosine filtering and up-conversion to IF.

Using 16 clock cycles per chip, one 'spreading engine' is capable of processing 16 physical channels. This block is then replicated to increase the total number of physical channels that can be processed simultaneously (currently 64).

The Inner Loop Power Control gains, TPC and FBI bits are applied last of all via a low latency link direct from the CPU.

### Layer 1 Management CPU

All the FPGA processes are managed by software on the CPU. Complex tasks such as rate matching pattern determination and Turbo interleaver parameter generation are performed in software leaving the FPGAs to perform mechanical operations.

## **CONCLUSION**

The software solution has traditionally provided greater flexibility over the committed hardware solution.

However, this advantage is rapidly diminishing, particularly in light of the steadily improving tools available for porting algorithms and system designs to FPGAs without necessarily resorting to hand-coded VHDL/VERILOG. Modern FPGAs now provide an enormously capable platform for implementing high speed / high complexity DSP that is every bit as re-programmable as a processor based solution.

Having completed a full implementation in software and a CPU / FPGA lab-prototype, the difficulties encountered with both methods lead us to conclude the following:

The multi-rate nature of the channel processing down to frame or slot rate benefits from the availability of a pre-emptive scheduler. The bit-oriented algorithms are very efficiently implemented in FPGA but are inherently not pre-emptable. This has the impact of making an FPGA 'co-processor' to assist a CPU a less than trivial option.

DSP processors are quite often very poor at bit-oriented processing; alternatives such as the PowerPC could be more appropriate.

The chip-rate processing (beginning at frame or slot rate) is relatively easy to perform in parallel for multiple physical channels and maps very well onto an FPGA.

## **REFERENCES**

3G TS 25.211 - Physical channels and mapping of transport channels onto physical channels (FDD),  
Technical Specification Group Radio Access Network

3G TS 25.212 - Multiplexing and channel coding (FDD),  
Technical Specification Group Radio Access Network

3G TS 25.213 - Spreading and modulation (FDD),  
Technical Specification Group Radio Access Network

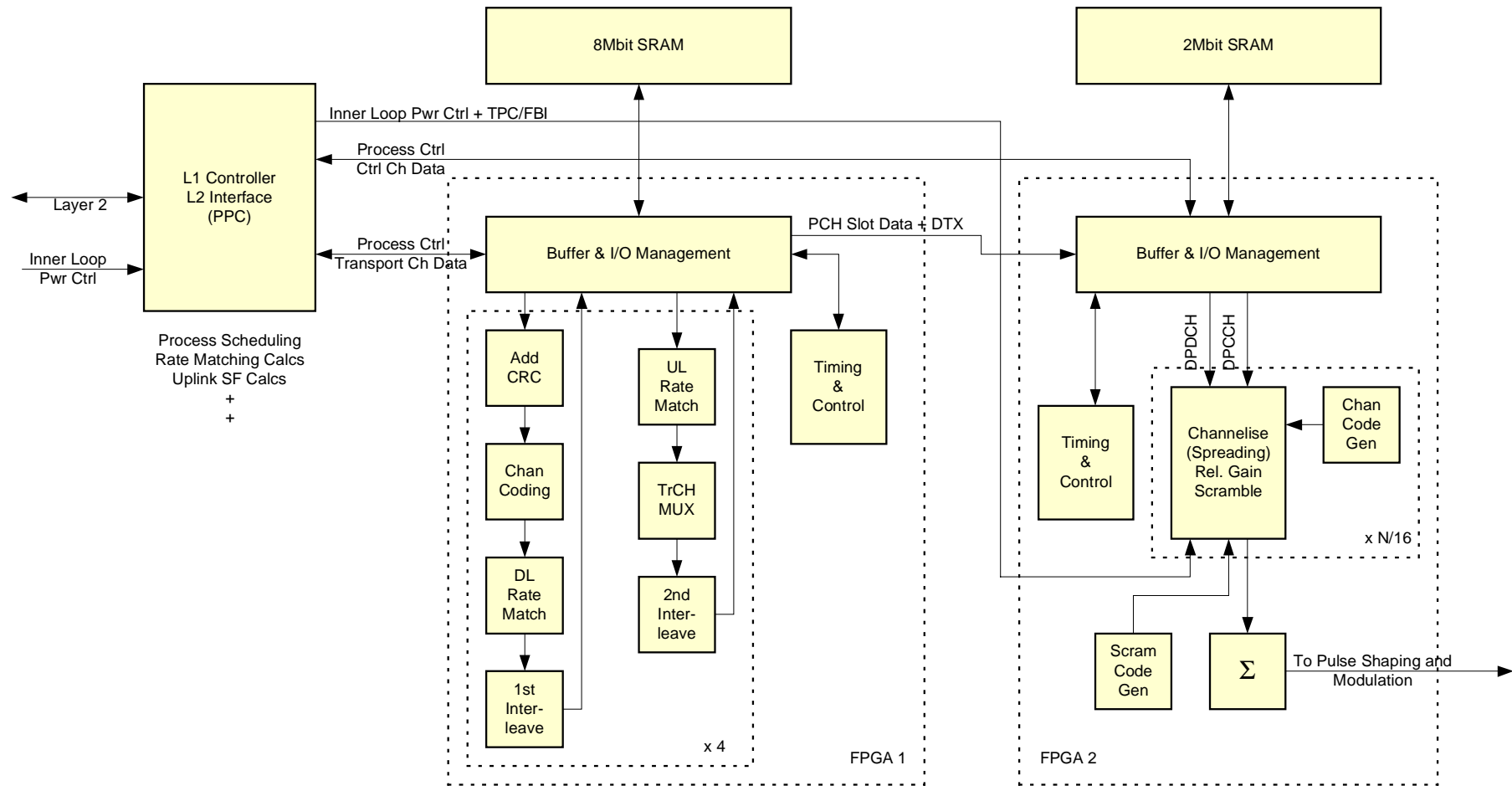


Figure 7: Experimental Hybrid CPU / FPGA Tx Path