A FEC CODEC-PROCESSOR (ASIP) FOR SOFTWARE DEFINED RADIO

- SoC – HW/SW – Co-design Experience and Analysis -

A. Blaickner (Carinthia Tech Institute-CTI, Villach / Austria, a.blaickner@cti.ac.at)
W. Scherr (Infineon Technologies, Villach / Austria, Wolfgang.Scherr@infineon.com

ABSTRACT

For multiple standard data communication purposes run-time re-configurability of the used air- and line interfaces is a preferred feature. Additionally to run-time re-programmable channel- and baseband-processing cores, a universal multimode forward error correction channel codec ASIP (application specific instruction processor) is a useful IP-core for application in software radio and storage applications. In this work, the concept and a prototype of a multimode codec processor ASIP was designed and verified. The required degree of flexibility and efficiency is attained by a pipelined Harvard like multi- processing architecture approach with dedicated hardware acceleration for the individual coding- and decoding tasks. The design is based on MatLab, C++, SystemC and HDL. Results on the design and the verification are presented.

1. INTRODUCTION

Multi-mode operation with increasing flexibility and performance figures are required in future digital communication systems and cellular networks. Additional constraints such as small area and power efficiency result in multiple tradeoffs for the mapping task of complex functions to a target software- / hardware-platform. A sufficient solution are so called ASIPs (*Application Specific Instruction Processors*) based on a pipelined architecture with multiple RAMs / ROMs and dedicated acceleration hardware which increases the processing speed of dedicated functional tasks. In this work the concept for a generic multi-mode forward error correction channel processor was designed and verified. The processor is based on multiple Harvard-architecture like control processor nodes (multi-processor-architecture) supported by dedicated arithmetic co-accelerators (ALUs) as e.g. Galois-field (GF) - adders and GF- multipliers, metric-computation, add-compare-select units (ACS) or survivor path selection (SPS). The individual processor nodes are pipelined or cross-connected over data path and control channels, so that a wide setup of run-time re-programmable Reed-Solomon and convolutional codec operations are available. Figure 1.1 shows the individual processing nodes with the attached dedicated hardware accelerators (ALUs). For example, the implementation of the Berlekamp- Massey – Algorithm (BMA) requires one processing node (PNs). The proposed solution provides sufficient flexibility for various codec operation setups, including code rates r or GF(x^n)-operations. For an example, see the Galois-Field processor architecture in detail in Figure 3.2.

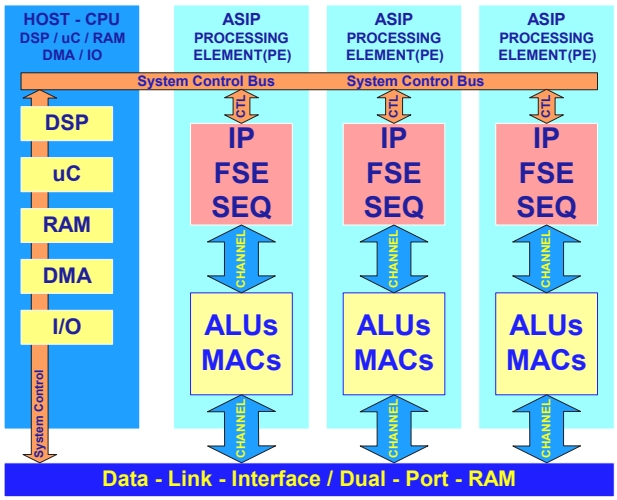


Figure 1.1: System acceleration by ASIPs

An overview of the design methods used is given in Figure 1.2.

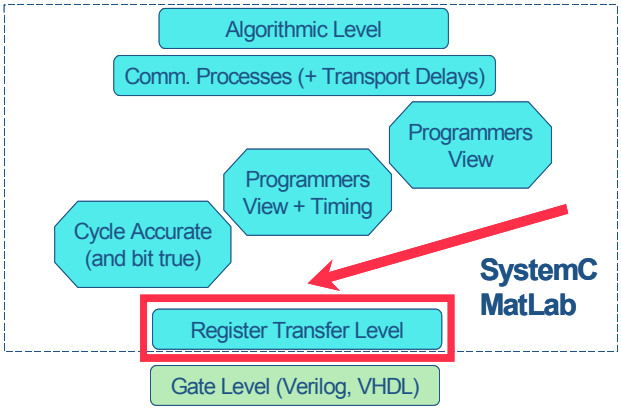
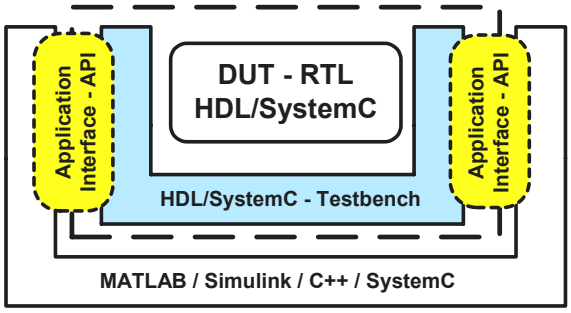


Figure 1.2: Design: MatLab – SystemC – HDL

The algorithmic, bit-true and cycle-true analysis are partially done either in MatLab or C++/ SystemC, the mapping to the implementation RTL- level either by a compiler / synthesizer or by VHDL-RTL-coding.

The design tool chain used for modeling simulation, synthesis and verification is based upon MatLab, Simulink, C++/SystemC (high level bit-true / cycle-true description), simulation of the DUT (behavioral / RTL), see Figure 1.3 is carried out with MatLab, ModelSim and GTKwave.



System Model - Overview

Figure 1.3: System Design Model

The analysis results and logic resources excluding the RAM space counted in gate equivalents (GE), which are required for the individual sections of the processor are depicted in Figure 3.5 and Figure 3.6.

2. THEORETICAL FRAMEWORK

The proposed error correction ASIPs cover the area of block-coding (Reed Solomon) and convolutional ML-decoding (Viterbi) and are foreseen in run-time reconfigurable SoC-, software radio or any data transmission/storage- application.

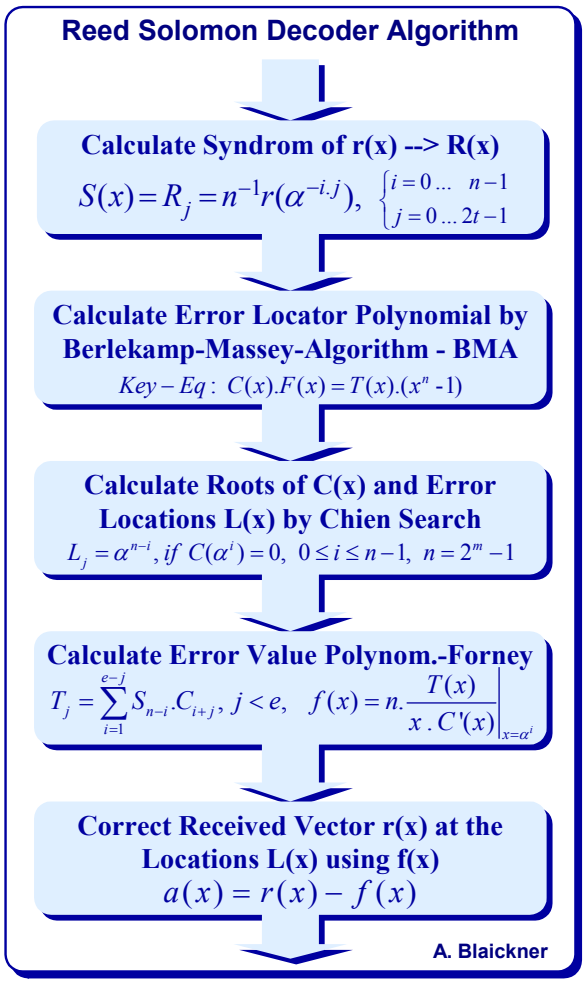


Figure 2.1: RS-Decoder – Algorithm

The algebraic decoding of Reed- Solomon codes can be hardware efficiently performed in five algorithmic steps as shown in Figure 2.1.

First of all, the so called Syndrome S(x) is calculated out of the received data vector r(x) by the DFT, which transforms r(x) into R(x). The Syndrome values S(x) can now be directly derived from the remaining 2E=n-k polynomial coefficients. In the next step one of the efficient algorithms – the Berlekamp-Massey-Algorithm (BMA) - is used for solving the so called Key- Equation which returns the error locator polynomial C(x). The BMA is an iterative method, with the principle shown in Figure 2.4. It searches for the shortest polynomial with coefficients C_i or otherwise the shortest feedback shift-register constellation using C_i, that is able to produce all frequency domain error vector components $F_0,F_1,\ldots,F_{2E-1}$ out of $S_0,S_1,\ldots,S_{e-1}$, with E (maximum correctable errors) and e (received errors). Next from C(x) the error location is derived by a brute force algorithm called Chien- Search, that inserts all possible values of x_i into the polynomial, computing the polynomial roots and their inverse values. Finally the error values f_j are calculated either by a recursive approach or by using the Forney Algorithm at the already derived error locations. In the last step the received data vector r(x) is corrected by adding f_j at the locations L_j. The derived solution for the computational most intensive part, the BMA-algorithm is discussed further in detail in chapter 3.

The performance of forward error correction systems can be remarkably enhanced using concatenated coding techniques, with block/cyclic- and convolutional codes combined. The concept of a convolutional decoder based on the Viterbi- algorithm was designed for ASIP implementation providing run-time reconfigurability options. An overview of the principles and basic operations required for convolutional decoding are shown in Figure 2.2 and Figure 2.3.

The convolutional encoding is based on generator polynomial operations represented by simple XOR and SHIFT register operations. The encoder calculates per k input bits n output bits at a code rate r=k/n using K bits (constraint length) from the shift register. The state transitions of the encoder are deterministic and show the behavior of a finite state engine, which is dependent on the current state and the new data inputs.

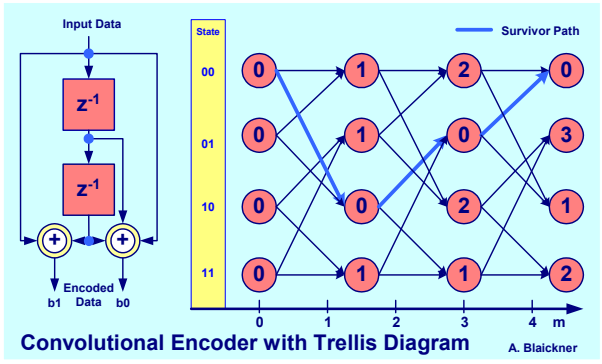


Figure 2.2: Example Encoder with the corresponding Trellis Diagram

After transmission and demodulation the decoder receives the encoded symbols including bit-errors. The Viterbi-decoder traces all possible paths through the state-sequence (trellis) and performs a maximum-likelihood search and decisions. The main computa-

tional units at the decoder are the Branch Metric Calculation Unit (BMU), the Add Compare Select Unit (ACS) and the Survivor Path Selection Unit (SPS), which can be seen in Figure 2.3. The BMU calculates a code distance metric λ_m between the received symbol and all reference symbols expected. This is done for all state transitions in parallel and provided as inputs to the ACS unit. The ACS unit accumulates the branch metrics λ_m and stores the resulting path metric γ_m, which is the actual sum of all branch metrics λ_m traced along the individual paths, see Figure 2.2. In radix–2 decoders the ACS is feed by two branch-metrics (r=1/n, with k=0) and the ACS does a decision for the most likely path, which is the minimum path metric $\gamma_{m+1} = \min(\gamma_m + \lambda_m)$ received and called best state decoding. Since the magnitudes for the path metrics are unbounded,

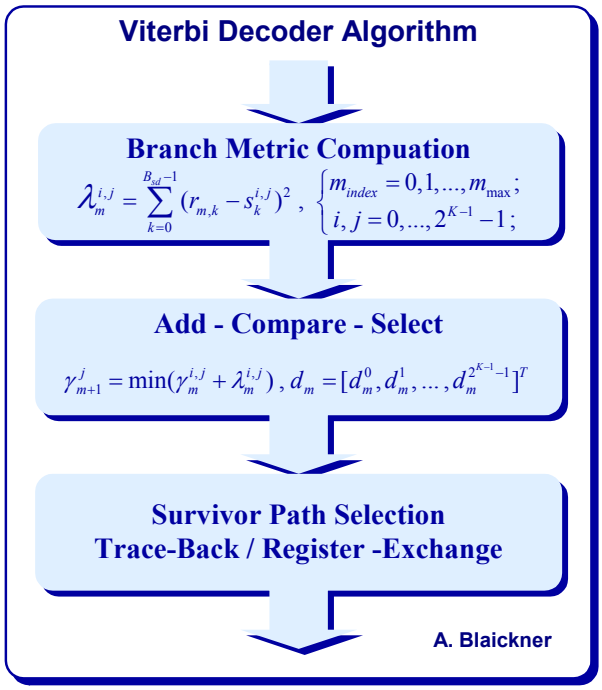


Figure 2.3: Viterbi decoder – Branch Metric Unit (BMU), Add-Compare-Select Unit (ACS), Survivor Path Selection (SPS)

one of the following normalization schemes need to be applied [9]:

- Periodic reset of the system to a ground state
- Use a redefined difference metrics ACS – algorithm
- Subtract the minimum metric from the survivors
- Saturation computation and block shift
- Modulo arithmetic scheme

The selected modulo arithmetic scheme avoids any kind of re-scaling, any data exchange between multiple ACS- units and shows advantages concerning hardware savings and speed-up of the inside metric update loop. As the path selection depends only on metric differences, it can be shown, that metric differences are bounded

$$\left|\gamma_m^0 - \gamma_m^1\right| \le \Delta_{\max} \le \lambda_{\max} \cdot \; ld(2^{K-1}) \qquad \text{Eq (1)}$$

with Δ_{max} is the maximum dynamic range of the path metrics required, $2^{(K-1)}$ is the number of states and λ_{max} is the maximum branch metric [10]. The required path metric precision is given then by

$$B_{width} = \left[1 + ld\left(\Delta_{\max} + \lambda_{\max}\right)\right] \qquad \text{Eq (2)}$$

due to the branch metric addition at the radix-2 ACS input, the term λ_{max} accounts for the potential dynamic range increase for the compare stage. Modifying the algorithm for positive metrics only, the precision required is B_{width} ,

$$B_{width} = \left[ld\left(\Delta_{\max} + \lambda_{\max}\right)\right] \qquad \text{Eq (3)}$$

with the decisions calculation as follows.

$$d_m = \left[\left(\left|\gamma_m^0\right| - \left|\gamma_m^1\right|\right) \ge 0\right]_{1/0} \oplus \left(\left|\gamma_m^0\right|_{msb} \oplus \left|\gamma_m^1\right|_{msb}\right) \qquad \text{Eq (4)}$$

For e.g. unsigned arithmetic, B_{sd}=3 bit soft decisions, λ_{max}=2*7, K=7 and Δ_{max} = 84, which results in a numerical bus width of $B_{width} = \log_2(98)$ or of B_{width} = 7 bit required at minimum.

After the decision bits have been computed in each ACS cycle, the decisions are stored into the decision RAM also called the Survivor Memory. Additionally the new calculated and selected path metric γ_{m+1} needs to be stored and is provided for the next ACS recursion (m+1).

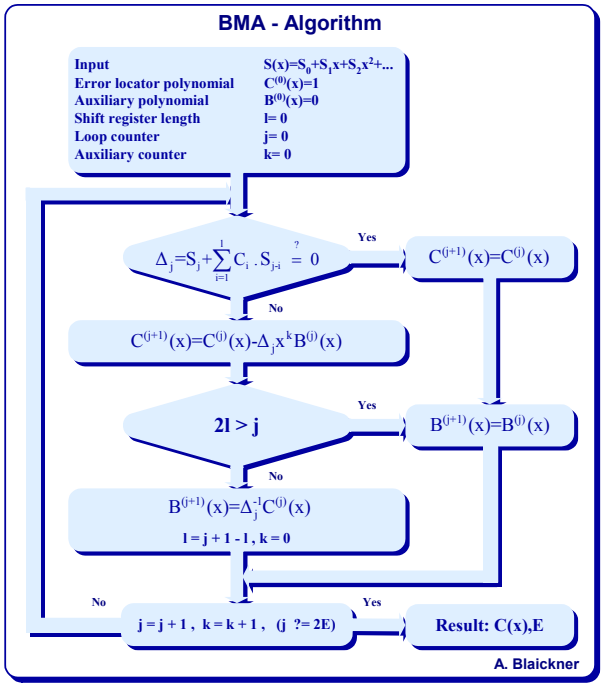


Figure 2.4: BMA-Algorithm

In order to recover the corrected data sequence two main methods, e.g. survivor path storage with trace-back or register exchange are available. Due to their hardware simplicity the first one was selected. After several traces (m = 10…m_{max}), the minimum path metric is identified and the trace back process is started. The trace-back unit makes use of the stored ACS-decisions (decision-RAM), a look-up table, which contains a corresponding state transition table (m → m-1) with the expected output data values. The previous state m with all actual decision d_m are held in two registers, which are utilized as address pointers either to select the

appropriate decision bit and to the state transition table. The final data value is directly output from the table look-up and stored into a data RAM.

3. FORWARD ERROR CORRECTION – ASIP

This section presents a more detailed excerpt of the implemented ASIP and PEs.

The basic algorithmic steps required for convolutional decoding based on the Viterbi- Algorithm have been already discussed in chapter 2. The concept and the model of the designed re-programmable multimode Viterbi decoder is depicted in Figure 3.1. The channel symbols are accessed from the input data storage and feed the Branch Metric Unit (BMU). Within the BMU, the required branch metrics (2^n, n-encoder output bits) are calculated out of the received channel symbol and the pre-stored references. To make use of more than one parallel processing ACS-units, the BMU also includes a switch matrix concept controlled over a lookup table, which allows the flexible distribution of all the branch metrics to each of the ACS- units. A similar approach is used for the ACS- processing and the data addressing required. The selectable degree of parallelism circumvents the ACS – bottleneck (required ACS-recursions for one trellis step m→ m+1) and increases the overall decoding speed.

The operations in the ACS are described in chapter 2. To ensure that the next ACS recursion is provided with the appropriate branch metrics, the ACS outputs are stored over an exchange network in the expected order. As described for decoding the final data sequence the Survivor Path Selection unit (SPS) makes use of the trace-back method with a selectable trace-back depth. The following model and run-time parameters can be selected:

- Maximum encoded output bits n
- Maximum constraint length K_{max}
- Bus-widths of soft-decisions
- Bus-width of system data paths
- Number of parallel ACS units
- Trace-back depth

The overall system is controllable by the usage of finite state engines and a host-interface / controller to setup the appropriate table contents and system modes.

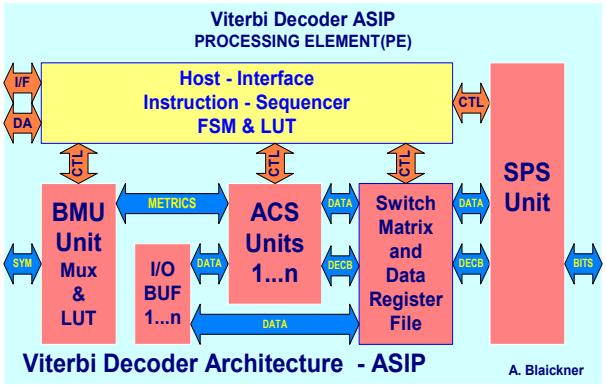


Figure 3.1: Viterbi Decoder architecture

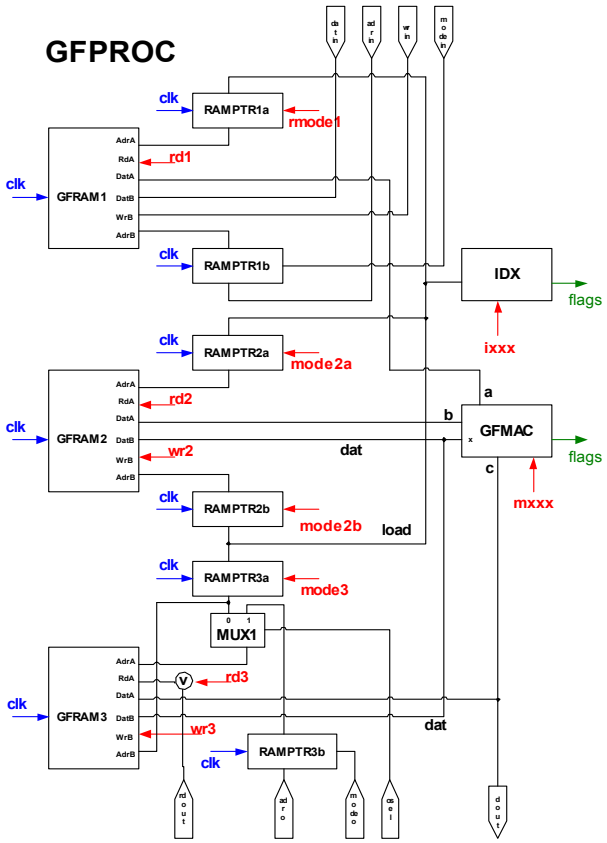


Figure 3.2: Galois field processing architecture

Next for the RS-decoder all the required Galois-Field-processing for e.g. the Berlekamp – Massey – Algorithm (BMA), the key – algorithm, is processed on one of the proposed processor unit as shown in Figure 2.4. The proposed solution provides sufficient flexibility for various kinds of codec operations and code rate setups. An example overview of the Galois-field processing element and its architecture is shown in detail in Figure 3.2 and Figure 3.3.

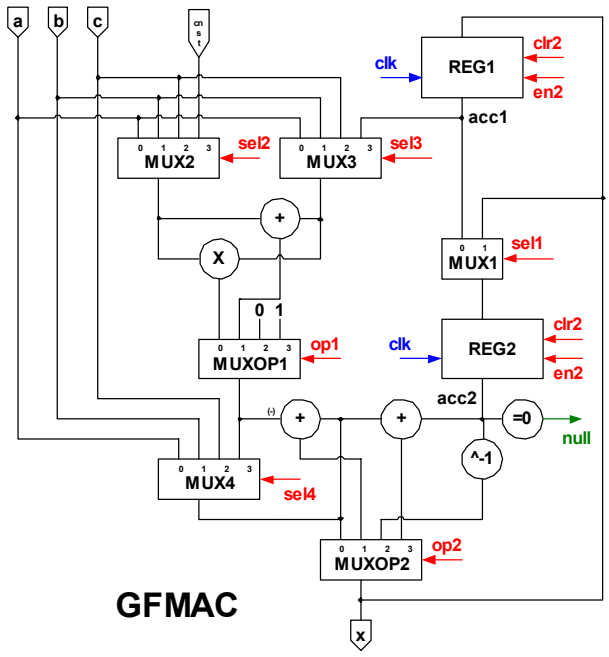


Figure 3.3: System acceleration with GF-MAC

The key features of the proposed GF processing unit are listed below:

- Optimized for fast data throughput
- Flexible by small programming ROMs
- Enables GF(n) and GF(2^n) operations with fields in the range GF(2^3) up to GF(2^8) by usage of flexible arithmetical functional units (AFUs)
- Supports GF - polynomials up to x^{255}
- Support of cascading and pipelined processing from an input channel (RAM) to an output channel
- Additional storage for intermediate data
- Provides conditional branches and polynomial index handling

The major blocks of the processing architecture are the processing-, the control- and the storage unit. The core processing unit and storage unit will be described in more detail. Firstly, we need to take a look at the field of operation for this unit. This is in this case handling Galois polynomials, which have the following form:

$$\alpha_1 \cdot x^0 + \alpha_2 \cdot x^1 + \alpha_3 \cdot x^2 + \alpha_4 \cdot x^3 + \ldots \qquad \text{Eq (5)}$$

For processing, especially the coefficients a_1,a_2,a_3, are of importance. So some memory to store these values in a useful way is required, including the positional index information. Furthermore these values are out of Galois- fields GF(n) or GF(2^n), where different arithmetical rules apply for calculations. When looking at typical polynomial calculations, some major operations can be identified:

- Sum of coefficients of one polynomial $\sum \alpha_n$
- Add/Multiply coefficients $\ldots + \alpha_n \cdot \beta_n \cdot x^3 + \ldots$
- Add/Multiply constants $\ldots + \mathrm{K} \cdot \alpha_n \cdot x^3 + \ldots$

These operations are required for a specific range or with different indices that may be also shifted from one to the other polynomial. Last but not least, those polynomials may be multiplied with any other polynomial and summed up again. Several combinations are possible and flexible data handling is needed. For these reasons, an index- calculation unit (IDX) was introduced. This unit can be used to increment or decrement indices, use them as loop counter, do some checking for greater or smaller value compared to each other and of course add or subtract different indices for shift operations within two polynomials.

A separate Galois-field calculation unit is provided, that accumulates, sums or subtracts input values. As two polynomial values may be summed up, a MAC unit is mandatory. To have the possibility to store an intermediate value, a second accumulator is added in the design. This allows also multiplying a constant and adding another constant to a coefficient in one single processing step. A detailed schematic of the MAC unit is depicted in Figure 3.3: , see also Ref. [5].

An example excerpt of the long instruction word assembly code for processing the Berlekamp-Massey-Algorithm within the RS- decoder (BMA) is shown in Figure 3.4:

```
+                          r r
+                         r m m r
+    i i          mmm m m mm o o m
+ i i s s iiiim m mmccs s s so d d o
+ao o e e llllo o eelle e e ed e e d wwrrr
+lp p l l ddddp p nnrrl l l le 2 2 e rrddd
+t2 1 2 1 43212 1 21214 3 2 13 b a 1 32321
:00000000000000000000000000000000000000000  000: nop (not really needed, just to test state)
:00010000001110000001100000001111111100000  001: k=l=j=0 acc1=acc2=0 ramptr1/2a/2b/3=0
:00000000000000001000011000000000000001000  002: ram2(0)=0
:00000000000000110000110000000000000010000  003: ram3(0)=1
:00011001000000000000000000000000001100000  004: ramptr1=m          % calc discrepancy first
:00011000110000000000000000000000000000000  005: n=l               % n is just loop counter
:00010000000000000000000000001100000000000  006: ramptr3=0
:00000000000000001100000000100000000000001  007: acc1=acc2=ram1(ramptr1) ramptr3++
:10011001100000001000000000001100000000000  008: JMP(n==0) 0x0c     % finished with summing up
:00001111110000000000000000010001000000000  009: n=n-1   ramptr1--  ramptr3++
```

Figure 3.4: BMA-decoder assembly code

Finally operational results of the implemented processing elements are depicted in Figure 3.5: .

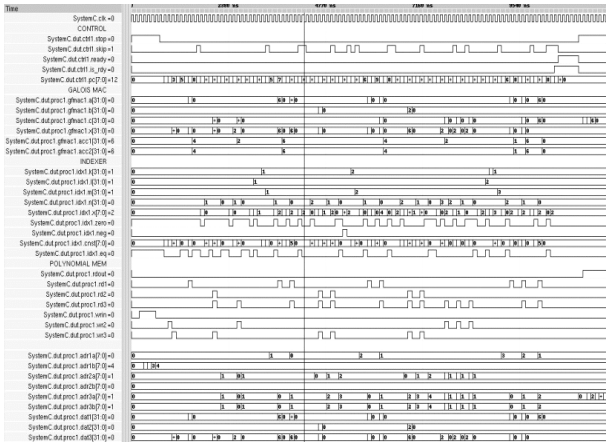


Figure 3.5: ASIP - Implementation results

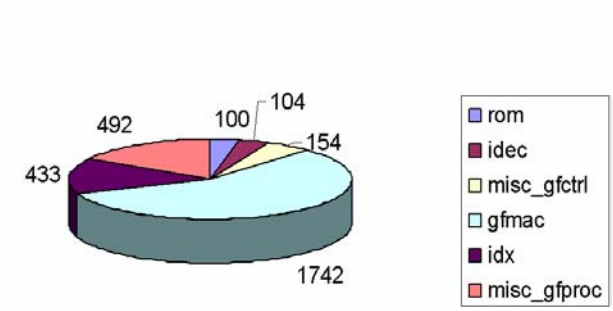


Figure 3.6: Logic resources required calculated in gate equivalents (GE)

4. DESIGN / IMPLEMENTATION PLATFORM

The design tool chain used for modeling simulation, synthesis and verification is based upon MatLab, Simulink, C++ / SystemC (high level bit-true / cycle-true description), see Figure 4.1 and the verification is carried out with MatLab, ModelSim and GTKwave.

The final synthesis step to the depicted DSP/FPGA platforms- as e.g. shown in Figure 4.2 is performed with Leonardo / FPGA-compiler / Quartus and ISE. The FPGA-prototyping platform PASS utilized a PC to DSP (Sharc 211xx) and DSP to FPGA – gateway approach which allows the real-time verification within the overall design-tool chain.

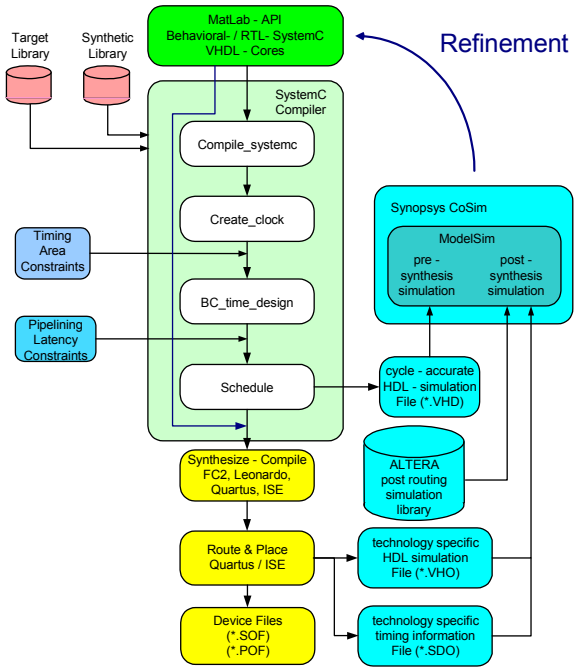


Figure 4.1: Design Flow - Tool Chain

Figure 4.2: Prototyping Platform – PASS

5. CONCLUSION

This paper presents the concept and prototyping of run-time re-configurable error correction coding ASIPs for typical application in wireless, software defined radios and data transmission / storage systems in general. The proposed and selected principles including the design methods with synthesis results were presented. Future work will focus on system enhancements and experimental setups with recent available high-level-system and behavioral architectural description-languages, providing guided and semi-automated system to architectural mapping and synthesis.

6. REFERENCES

[1] Peter Sweeney, “Errors Control Coding – An Introduction”, 1991.

[2] A. Blaickner, H. Sterner, M. Bacher, Liu Shih-Fu, “A Software Defined Radio Channel-Processor for 3G-Systems - SoC – Design Experience with SystemC, MatLab and VHDL”, in Proc. of Software Defined Radio Technical Conference – SDR ’03, Nov. 2003, Orlando, USA.

[3] R. Blahut, „Information Theory“, Addison Wesley, 1988.

[4] A. Blaickner, J. Madsen, H. Holten-Lund, M. Bacher " Design of a Multi-Mode -Channel- Select and Re-sampling-Processor (ASIP)", in Proc. GSPx 2004, Sept. 2004, Santa Clara, USA.

[5] Thomas Richter, Gerhard P. Fettweis, “Parallel interleaving on parallel DSP architectures”, 2002.

[6] W. Kester, “Mixed-Signal and DSP Design Techniques”, 2003.

[7] S. Lin and D. J. Costello, “Error Control Coding: Fundamentals and Applications”, Englewood Cliffs, NJ: Prentice-Hall, 1983.

[8] Ungerboeck and H. K. Thapar, ”VLSI Architectures for Metric Normalization in the Viterbi Algorithm,” Proc. ICC 90, Vol. 4, Apr '90.

[9] A.P. Hekstra, ”An Alternative to Metric Rescaling in Viterbi Decoders”, IEEE Trans. Com., Vol 37, No. 11, pp. 1220-1222, Nov 1989.

[10] P. H. Siegel, C. B. Shung, T. D. Howell and H. K. Thapar, ”Exact Bounds for Viterbi Detector Path Metric Differences”, Proc. ICASSP 91, May 1991.