

• Timing constraints :

The input data rate is fixed on 10Mbyte/sec. The tolerated latency is $3N + 6T$ clock cycles, with a system clock of 10 MHz.

Those constraints define the design space to be explored in order to find the optimal solution.

III. Design Flow

The first step in the design flow is to describe the function that needs to be implemented. Next is a behavioural simulation phase which verifies the description against the given test stimuli and responses. Note, that the description at this point is purely functional and not yet committed towards a certain implementation style.

At this point a decision must be made about the hard- or software partitioning of the system. Parts of the system can be implemented in assembler for commercial DSP cores, parts can be synthesized onto an ASIC. In this project, it was clear from the beginning that the complete system had to be mapped onto an ASIC because of high speed requirements.

The path towards the ASIC implementation starts with partitioning the design into pieces which can then be implemented with the most suited target architecture at hand. Currently no real automated tools exist to partition complex applications like this one, so this was done mostly based on human reasoning.

Then next design phase is the architectural design phase. The goal of this phase is to explore the design space, to evaluate different architectural alternatives and to generate a VHDL netlist of the IC in terms of RAM's, ROM's, ALU's, The architectural design phase is supported by the Mistral 2 True Architectural Synthesis system of Mentor Graphics and the Cathedral-2/3 architectural synthesis system of Imec which is tightly coupled with the DSP Station design database via a procedural interface. The Mistral 2 architectural synthesis system provides an implementation path towards microcoded processor architectures and the Cathedral- 2/3 synthesis system provides a path to highly pipelined, data path dominated processors. The final VHDL netlist can be verified against the (bit-true) behavioral simulations and is input to a logic synthesis tool.

During logic synthesis, the netlist is translated into gates and mapped towards a technology. After simulation at this level, place and route tools are used to generate the layout.

Thanks to those different phases, the designer can concentrate successively at different design parameters. The architectural design phase focuses on the composition of the data path and the global cycle count. Logic synthesis performs local optimizations in terms of hardware and timing and the place and route tools compose the final layout. Each phase is to some extent independent of the other ones : the behavioral specification phase is implementation independent. The designer still has the freedom to opt for a software implementation or a hardware solution. If he chooses for the hardware alternative, he still has the choice between different target architectures. The architectural design makes as much as possible abstraction of the target technology details. Finally, the result of the logic synthesis still can lead to different layouts.

The benefit of this design environment is mainly located in the automation of the architectural design phase that allows for fast architecture exploration and evaluation. The design decision on certain architecture trade-offs can now be based on real evaluation results rather than on abstract reasoning and estimation.

Behavioral specification

The complete algorithm has been described in DFL, a dedicated signal processing language developed in cooperation with UC Berkeley [2]. DFL is a language that allows to make a **behavioral specification** of the algorithm which does not impose restrictions on the next phases in the design process. The behavioral specification of the algorithm is independent of the implementation target , as well in terms of hardware as in terms of timing aspects of the design. For example, the presence of a multiplication in the algorithm description, does not automatically imply the presence of a multiplier in the data path. The designer may, during the architectural design phase, possibly opt for a sequential implementation of the multiplication by means of a booth algorithm on an ALU. Moreover, the order of the statements in the algorithm description does not influence the scheduling of the associated register transfers on the data path. The specification phase becomes far too complex if scheduling aspects have to be taken into account at the moment one has to be able to concentrate only on the behavioral correctness of the description.

DFL is an **applicative language**. This means that the order of statements does not influence the behavior. Only data flow dependencies between input and output determine the sequence of executions. This property of DFL is an important differentiator with respect to procedural languages such as C. Another strength of

DFL is its ability to efficiently describe **finite word lengths**. As a hardware implementation most often implies a finite bit width representation of the signals, it is necessary that this finite word length characterization and evaluation can be done at the highest level. Only the detection at the specification level of unwanted bit-true effects can prohibit the design of a non working chip due to overflow or quantization effects.

In order to unambiguously fix the specification, a coder-channel- decoder cascade has been described. The channel function allows to introduce any number of errors and erasures. Besides, in order to be able to operate in the Galois Field GF(256), a library of user defined functions is written in DFL to perform the GF addition, multiplication and division.

The global DFL description is hierarchical and frame based. Parameters of the description are N (code length), T (error correcting capability), v (number of errors), and p (number of erasures). The final specification contains 500 lines.

Behavioral simulation

The next step in the design flow is the behavioral simulation. The purpose of this task is twofold. On one hand, the behavioral correctness has to be checked and on the other hand the bit-true behavior has to be verified. Starting from the DFL description, a SFG (Signal Flow Graph) is automatically generated. This SFG is the starting point for both the time domain simulator and the architectural synthesis tools. The time domain simulator can operate in two modes : high level and bit-true. Both modes start from one and the same SFG, i.e. one and the same DFL description. In the high level mode, the simulator treats all signals as being floating point signals with infinite accuracy. In this way, only the behavior is checked without having to worry about bit-true effects. By probing intermediate signals, the description is debugged until a correct behavior is accomplished. Starting from this behaviorally correct DFL description, the time domain simulator is invoked in the bit-true mode. This simulation step allows the designer for checking whether finite word length effects cause the description to behave differently.

For both modes, the parameters can be changed interactively, without having to recompile the DFL. A simulation time of 1 minute per input frame proves that the debugging can be done very efficiently.

The specification and simulation phase required 1 man month for the complete Reed-Solomon codec.

The approved DFL description is translated into an SFG, which forms the input for the synthesis tools Mistral2 and Cathedral-2/3.

Partitioning Strategy

Once the behaviour is fixed a decision must be made about the different hardware partitions. This partitioning is done by hand, based on the following observations :

According to the system specifications, a new data word is coming in each cycle of the system clock. This high throughput requirement, combined with the computational complexity of the algorithm, indicate that the ASIC implementation must be a pipelined structure. The algorithmic partitioning is based on data flow considerations between the different functional modules. A first section is defined by the encoder (P1) which has a small computational complexity. The decoder can be subdivided into four major functional blocks:

- the syndrome generator that calculates the syndrome of the received code word
- the Berlekamp-Massey algorithm that computes the coefficients of the Error Locator Polynomial (ELP) and the Errata Evaluator Polynomial (EEP) and the Premultiplication block that will enable the next pipeline section to treat the erasures in the same way as the errors
- the Chien algorithm and the Errata Value block to compute the errata locations and values
- the correction block which computes the syndrome from the corrected RS code. If the syndrome is zero, the chip outputs the corrected RS code, otherwise the chip outputs the received RS code with ALARM signal

Besides the conceptual partitioning, other criteria have to be considered to partition the algorithm for hardware implementation. These are mainly the performance of the target architecture and the interfacing constraints. The target architecture of each partition is determined by its algorithm characteristics. Highly regular computation schemes with a low level of conditional actions require a regular data path structure on which control tasks are hard coded. Algorithms characterized by a lot of looping and decision making, are suited to be mapped to a data path controlled by a microcoded controller.

A good partitioning groups these parts of the algorithm that map to the same target architecture. During this step, the computational load is also taken into account in order to obey the latency constraints. The ease of interfacing, determined by the data passing and synchronization requirements, significantly influences the complexity and size of the hardware that takes care of the communication between the different

modules. The conclusion is that during the design partitioning phase, the designer tries to find the optimal solution which balances between the system requirements, the algorithm characteristics, the size of the hardware and the complexity of interfacing between the partitions. Based on these arguments, the Reed-Solomon algorithm is partitioned as shown in figure 2.

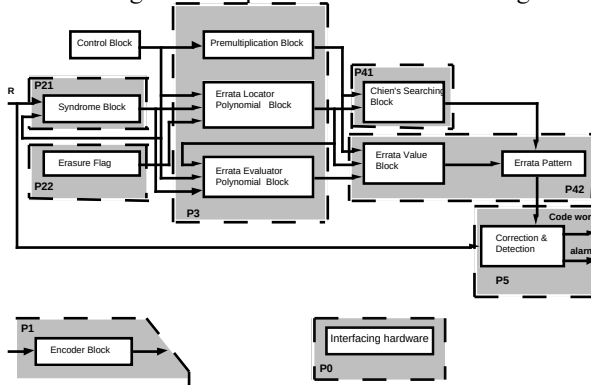


Figure 2 - Design Partitioning of Reed-Solomon Codec Algorithm.

The first partition P0, consists of communication and synchronization hardware between the different pipeline sections. This partition is manually designed and mainly contains a FIFO with some controlling logic around. Next, there are a number of blocks which are characterized by the same algorithmic properties, i.e. regular computation scheme and low level of decision making. These include the encoder, the syndrome generator, the erasure flag generator, Chien's searching block, the errata value calculator and the correction and detection block. Chien's searching block and the errata value block can share their hardware as they are both evaluating polynomials. However, to be able to meet the latency and throughput constraints it was decided to provide parallel hardware. This results in the hardware partitions P41 and P42. Functionally, the encoder is well suited to be processed on the same hardware as the syndrome block. The system specifications require the ASIC only to work in half duplex mode, which means that the chip is either encoding or decoding. Hence, because both modes do not need to be active simultaneously, the encoder can share its hardware with the syndrome block.

The control flow of the Berlekamp-Massey block depicts a higher complexity than in the other algorithmic partitions. Therefore, it was decided to opt for a microcoded architecture to implement this functionality. As the data path of the Berlekamp-Massey block contains 20 multipliers, it was decided to share these multipliers to execute the Premultiplication function. As a consequence Berlekamp-Massey and Premultiplication functions share the same data path hardware (P3).

In order to meet the speed requirement, pipeline registers have been added between

- partitions P21 and P22 and partition P3
- partition P3 and partitions P41 and P42
- partitions P41 and P42 and partition P5

Implementation of the different partitions

Based on the characteristics of the different partitions as described above the Cathedral-2/3 compiler has been used to synthesize the lowly multiplexed data paths that execute respectively:

- the syndrome block and encoder block
- the Chien's search block
- the error evaluator block
- the alarm detection block

The Berlekamp-Massey and Premultiplication block are synthesized as one hardware block by the Mistral2 compiler.

The input for one Cathedral-2/3 design thread are different signal flow graphs, which are executed under exclusive conditions, i.e. they are not executed in the same time interval neither under identical control conditions. The signal flow graphs are automatically generated from the DFL description of the algorithm. These exclusive signal flow graphs are executed on the same data path. The netlist of the resulting multi-functional data path unit (Application Specific Data path or ASU) and the associated local controller is generated by Cathedral-2/3. Afterwards, the netlist is passed through a retiming tool which performs a timing analysis and adds, if necessary, internal pipeline registers. The output is either a hierarchical or a flattened VHDL netlist.

Syndrome computation and encoding block

The syndrome computation and the encoding are combined into one circuit. During decoding mode, it calculates the syndromes and during encoding, it performs the encoding of the message. The syndrome block in partition P21 calculates 2T syndromes which are the frequency components of the received word. If the received word is a correct codeword, then all syndromes are zero. Otherwise, the syndromes are used to determine the error locations and values. The syndrome generator can be realized as shown in figure 4. The hardware needed for this block consists of 2T Galois field adders, 2T Galois field multipliers and 2T

registers. Since T is ranging from 1 to 10, 20 taps are required with element multipliers going from 1 to 20. For a certain value of T , the first $2T$ taps of the circuit contain the syndromes.

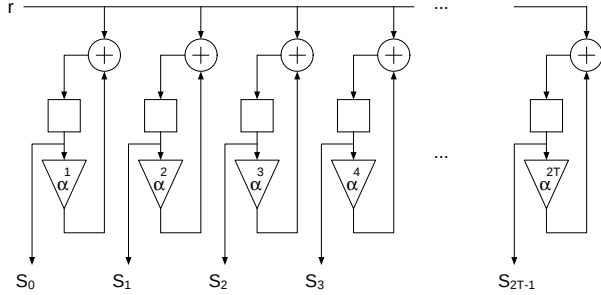


Figure 4 - Hardware realization of the syndrome generator

The encoder block adds $2T$ to the K incoming message symbols, thus producing a codeword of N symbols. The incoming symbols are left unchanged because a systematic code is used. An encoder can be realized in a classical way by multiplying the message symbols with the generator polynomial. The synthesis of this combined circuit was done by writing a DFL description for the encoder and the syndrome generator for each value of T . The 20 resulting signal flow graphs are fed to Cathedral-2/3 which generates a data path that is able to calculate each of the signal flow graphs. The generated data path contains GF adders, GF element multipliers and two input multiplexers. The multiplexers are needed to disconnect the taps from each other during syndrome calculation and to be able to reset the registers. Note that this is a typical high throughput application since 20 GF additions and multiplications have to be performed per symbol.

Berlekamp-Massey

Partition P3 executes the Berlekamp-Massey algorithm, together with the premultiplication block. The decision making characteristics of the algorithm and the throughput requirements make this partition suited to be designed with Mistral2.

The computational load of this pipeline section is quite heavy, since the module needs $84T + 20$ Galois field multiplications, $76T + 100$ Galois field additions and 1 Galois field division. The maximum allowed latency is $3N + 6T$ cycles of the system clock.

By default, Mistral2 will generate a data path containing units as an ALU, a Multiplier, a RAM, etc. However, the designer would like to make use of hardware units which are tuned to the application in terms of area and/or speed. In this project, the designer is obliged to use ASU's. The default ALU and Multiplier are not able to perform arithmetic in the

Galois field. Moreover, the combination of the latency constraint and the heavy computational load require application specific hardware (ASU).

Although the Berlekamp-Massey algorithm operates in the finite Galois field $GF(256)$, there are still some two's complement operations to be executed. The LNASU takes care of this arithmetic. It is able to compute the maximum of two 8 bit numbers and to evaluate conditions. All inputs and outputs are two's complement 8 bit integers.

The INVASU calculates the inverse of an $GF(256)$ element. This $1/x$ behavior is implemented as a lookup table. The MULTASU on its turn is just a Galois field multiplier on 8 bit integers

However, the kernel of the Berlekamp-Massey partition is the DKASU. This block calculates the coefficients of the Error Locator Polynomial and the Error Evaluator Polynomial. In order to meet the latency constraint, this ASU has to contain multiple Galois field multipliers (as defined by the MULTASU) and Galois field adders. As the computational schemes for the calculations of both polynomials are different, extra control logic is needed, which has also to take care of the correct execution of the premultiplication function.

All those ASU's are designed with DSP Blocks. This tool allows to describe a netlist within a schematic editor, making use of elements of Mentor Graphics LogicLib. For each of those parameterized library elements, an instantiated VHDL model can be generated. DSP Blocks generates then a VHDL netlist for the complete schematic. In this way, a VHDL model is derived for the four ASU's. The interconnection between the different ASU's and the control flow of the Berlekamp-Massey partition is generated by means of Mistral2. The data path contains, besides the 4 ASU's, a 5 bit ACU. The global controller is programmed with 20 64-bit micro instructions. The time to propagate an input to the output of the data path amounts to $9T + 8$ cycles of the system clock. In this phase of the design, the frequency of the system clock does not matter, since the architectural design is independent of the technology. The fact that the very severe timing constraint could be met is not only a consequence of the fact that DFL retains the inherent parallelism of the algorithm. The scheduler of Mistral2 allows for automatic loop folding. In this way not only operations within the same loop iteration can be executed in parallel, but also operations belonging to different loop iterations can.

The architectural exploration and evaluation, the generation of the final VHDL netlist at ASU level and its simulation, required all together 2 man months.

Chien's search and error evaluator block

The next block in the correction procedure is to compute the roots of the premultiplied error locator.

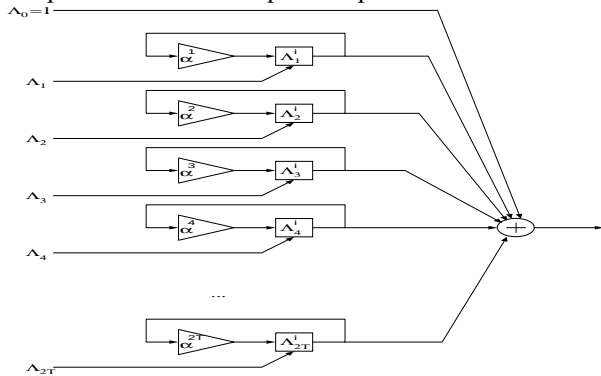


Figure 6 Iterative evaluator for all field elements

Since the Galois field is finite, the easiest way to find the roots is to enumerate them all and try them one by one. However, Chien's search algorithm provides an iterative method that avoids field multipliers and that uses $2T$ element multipliers instead. Figure 6 shows a way of iterating over all field elements. Each frame clock, a new field element is tested to be a root. If the output is zero in the i -th iteration, α^i is a root.

Alarm detection

The alarm detector computes the syndrome of the corrected symbols. If not all the syndrome components are zero, an alarm signal is given, indicating that the corrected message is not a code word. This means that the numbers of errors introduced in the encoded message has exceeded the error correcting capabilities of the code. The alarm signal can then be used to request a retransmit. If the alarm is set, the uncorrected message is passed to the output instead of the corrected message. The implementation for this block consists of a syndrome generator based on the same structure as described above, followed by a circuit that checks if the first $2T$ syndrome components are zero. For the synthesis, this was split up in two data paths.

The **first** data path calculates the syndrome and checks for each syndrome component if it is equal to zero. It is generated by Cathedral-2/3 starting from a DFL description for $T=10$. This circuit has 20 boolean outputs. the hardware needed consists of multiple GF adders, GF element multipliers, comparators and two input multiplexers to realize the resetting of the registers. The **second** data path calculates the alarm signal by checking 2 of the 20 outputs that are calculated in the first data path. This circuit is described

by a truth table and synthesized by means of logic synthesis. The area is negligible compared to the first data path.

Interconnecting hardware

For all pipeline sections a VHDL netlist is generated in terms of Galois Field arithmetic operators, memory devices and local controllers. The next task is to define the interconnecting hardware. For this purpose, a VHDL model of a FIFO is written. As the design is parameterized in terms of the code length N and the error correction capability T , the read and write pointer of this FIFO have to be presettable according to the values of those parameters. Extra logic is foreseen in order to trigger the "Start of Frame" flag of the different partitions after each $3N + 6T$ cycles of the system clock.

Global design results

The gate count of the complete design is 100,000 gates. One third of these are assigned to the Berlekamp-Massey partition. The design time is reduced from four man year (classical way of design) to two man year. At this moment, the chip is being processed.

V Conclusions

This paper has demonstrated the use of high level synthesis techniques for the realization of a high speed Reed-Solomon Codec. The functionality of the codec and the different design trade-offs have been explained. The design was realised with a combination of DSP Station from Mentor Graphics and the Cathedral-2/3 compiler from Imec. The latter compiler is coupled to the DSP Station data base via a procedural interface. Currently, the functionality of Mistral2 and Cathedral-2/3 is being combined in a single compiler. This will result in the possibility to perform design exploration over a large variety of architectural styles ranging from medium speed micro-coded processors to highly pipelined, lowly multiplexed data paths.

VI Bibliography

- [1] R.E. Blahut, "Theory and practice of error control codes", Addison Wesley, 1983
- [2] P. N. Hilfinger, "A high level language and silicon compiler for digital signal processing", IEEE Custom Integrated Circuits Conference, Portland, OR, , 1985
- [3] S. Note, F. Catthoor, J. Van Meerbergen, H. De Man, "Definition and Assignment of Complex Data Paths for High Throughput Applications", IEEE International Conference on Computer-Aided Design, Santa Clara, California, , Nov 1989

