Fast Low Complexity Reed Solomon Codec for Space and Avionics Ramdisk Applications

M.S. Hodgart and H.A.B. Tiggeler

Surrey **S**pace **C**entre,

University of Surrey,

Guildford, Surrey, GU2 5XH, UK.

Tel: +(44) 01483 259278

Fax:+(44) 01483 259503

Email: S.Hodgart@ee.surrey.ac.uk , H.Tiggeler@ee.surrey.ac.uk.

http://www.ee.surrey.ac.uk/EE/CSER/UOSAT/

Abstract

We describe an application of the RS code to the routine error protection of large RAM memory for satellites in low Earth orbit. Errors are normal in this environment - an estimated 1000 soft errors per day for 128 Mbytes of memory being typical. There is a need to free up the CPU from most or all of the required encoding and decoding, and use a dedicated codec. The device is mostly transparent to the routine transfer of data between CPU and RAM disc Our choice of codec uses FPGA technology. The application requires correction of a maximum of two (byte) errors per block of 512 bytes of data. This limited number of corrections obviates the need for standard decoding methods. A form of direct decoding is adopted that allows very considerable simplification of the codec with a low internal clock rate and low physical complexity

1. Introduction

Reed-Solomon error correction codes are well-known and widely used. They combine powerful error correcting properties with a low overhead in terms of redundant coded data. A perceived disadvantage is the relative complexity of the decoding process [1]. The theoretical difficulty may have conditioned the engineering community to rely exclusively on standard methods. These methods are powerful but indirect, requiring extensive computations [2] [3].

If the code is restricted in performance then it is possible to design a Codec of greatly reduced complexity. We refer to this option as *direct decoding*. Our restriction is to a $t = 2$ (distance 5) RS code that will decode up to *two* error words per block of data (DEC), but no more than that.

The RS codes are a family of codes. The higher members of this family are capable of better correction performance for which direct decoding is not applicable and the standard methods must still be used.

In space applications it is well known that in low Earth orbit (LEO) stored digital data suffers from Single Event Upsets (SEUs). These upsets are induced naturally by radiation. **S**urrey **S**pace **C**entre (SSC) has experience over 17 years of data handling on micro-satellites in LEO. We have conducted a measurement campaign of radiation induced errors [4]. From this experience one can estimate 10^{-6} SEUs per bit per day. The memory is not damaged but the data is corrupted on every SEU. It is also our experience that 99% of these SEUs affect only one byte of data (and usually just one bit of that byte). The remaining 1% affect two bytes. It is rare to get three byte errors (quantifying these rare events is not easy)

Up to now we have been using a single-error correcting and double error detecting (255,252) ×8 RS code to protect large blocks of static memory against these radiation-induced SEUs. The encoding and decoding process is executed entirely in software.

The largest memory size using soft RS coding has been 16 Mbytes. The memory requirement for our latest generation of on-board computers is however an order of magnitude greater. A size of 64 or even 128Mbyte of static memory per computer is quite common or will be (see figure 1).

For the largest size of memory we can expect 1000 SEUs per day per computer. The encoding and decoding of data is therefore a *necessity*, to be applied on every transaction between the Ramdisk and the controlling CPU.

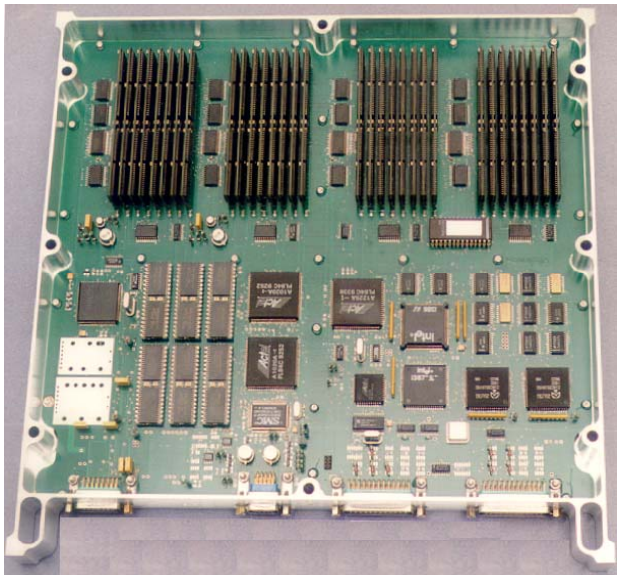


Figure 1 UoSAT Secondary 386EX based On-Board Computer with 128Mbyte Ramdisk

A software implementation would have a drastic effect on overall processing power of the CPU. Studies performed using QNX operating system on a 17MHz 386EX show a file system throughput degradation of nearly 80% (see table 1). For present and future requirements a hardware implementation of the RS code - a codec- is also seen to be a necessity.

Operation	Read [Kbytes/Second]	Write [Kbytes/Second]
Raw Bus Bandwidth	**3780**	**3780**
Without Reed Solomon Coding	200	160
With Reed Solomon Coding	**41**	**28**

Table 1 QNX File System benchmark 17MHz 386EX

2.Codec description

Our operating system for the CPU adopts an invariant block of 512 data bytes when writing to or reading from the Ramdisk. This is our unit of data and is to be encoded/decoded as a whole.

From the RS family of $t = 2$ correcting codes we chose one of natural size (1023,1019) × 10. This code is then *shortened* and *reduced*. The number of data bytes is shortened to 512 while each byte is reduced from the natural 10-bit wide word of this code to 8 bits.

Encoding

As an encoder the Codec reads the 512 bytes of data and treats each byte as a 10-bit data word having 2 virtual zeroes (see figure 2).

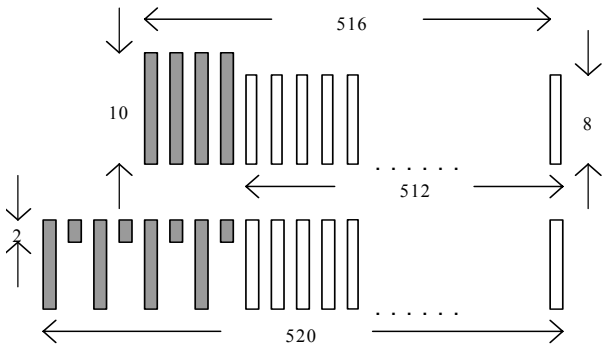


Figure 2. block format before/after partitioning

The encoder generates 4 × 10- bit parity words per block. Each parity word is partitioned as 8 + 2 bits over two bytes - generating 8 additional parity bytes to be appended to the block of data and stored with the data. The overall construction may be termed a hybrid RS (520, 512) × 8 code. In this implementation the increase in memory requirement is a negligible 1.6 %.

Decoding

As a decoder the codec reads all 520 bytes of data, and re-packs the parity bits back to 4 × 10-bit words. Internal processing within the codec is of 516 words that are 10-bit wide (to the codec)

We define a stage-1 of error detection. This is a calculation of 4 10 bits wide *syndromes*. If all these syndromes are zero then no error has been detected (normal case). If one or more syndromes are non-zero then an error has been detected.

A stage-2 then seeks to identify and locate errors that have been detected. It processes the 4 syndromes from stage-1: at the conclusion either (i) one error byte has been identified and located (ii) two error bytes have been identified and located

The codec is able to do more than identify and locate (up to) 2 error bytes however in any two random locations in the block. There is a beneficial result of constructing this reduced and shortened RS code: it is capable of extended-error detection (EED). Random tests show 97% success for the full block size of 512 bytes in detecting 3 or more random errors. If this judged insufficiently reliable the success rate can be increase simply by reducing the block size, with the cost of increasing redundancy. There are other options.

A stage-3 would complete the decoding process by correction of the erroneous block of data

code computations

The encoding and stage-1 decoding (deriving the syndromes) are standard processes and known for the relatively simple implementation

Coding theorists know that the problems lies in stage- 2: traditionally requiring first the determination of an error location polynomial and processing of this polynomial (Berlekamp Massey or Euclidean algorithms)

We have shown however with $t = 2$ requirement there is no need to follow the standard approach. We go back to historically early *direct decoding* that has a low complexity

One measure of complexity of calculation is the number of multiplications of Galois field elements per processed data byte.

For the implementation of an 'n-k' encoder requires four GF multiplications per byte. These are *fixed multipliers* and lend themselves to modern software logical reduction. In our implementation the required GF calculation of summation and multiplications are done in one step i.e. are fully parallel.

The encoder may be stepped at the same rate as the data, with the required parity bytes available immediately after the last data byte.

The standard stage-1 decoding requires three GF simple *shift*-multiplications per byte to realise the four required syndromes (these are very simple objects).

With direct decoding stage2 is not much more complex than stage1. To compute the error values requires a *squarer* and three GF shift multiplications per byte and one *general multiplier* per processed byte over 512 steps. To locate the errors needs a squarer and one general multiplier. This take a further 256 steps. Other than this only EXOR summations are required. The highly complex *inversion* of GF field elements is *not* required.

The step rate (within the chip) is twice the basic clock rate and therefore the codec is able to complete stage-2 decoding within one frame time. The chip performs stage-2 decoding simultaneously with stage-1 decoding of the next block. There is therefore just one block delay in reading from memory to the CPU in case of an error.

Codec options

We define three practical options:

option 1

This implements encoding and stage1 decoding only: the Codec passes back the syndromes only, assuming a software routine in the CPU exists to complete the decoding. We may note that the software computation has to be done only when there has been a hardware detection of an error and at no other time. Although 1000 error events per day may seem a lot, it is a very small fraction of the total read/write transactions. This option makes the least demand on Codec complexity as measured by the lowest gate count implementation (see table 2). So with just option 1 there has been a substantial shift of computational effort from software to hardware.

option 2

This implements encoding, stage-1 decoding and stage-2 decoding. The codec passes back the identity of one or two error bytes and their location to the CPU. It also signals an EED (detected but uncorrectable error) back to the CPU.

option 3

This was not seen as a viable option because it is necessary to store the entire block of data. At the present time no Radiation Tolerant FPGA is currently available that is able to support the large number of required RAM cells. Such cells would need Hamming code protection to remove their susceptibility to radiation. This would increase the required memory by nearly 50%.

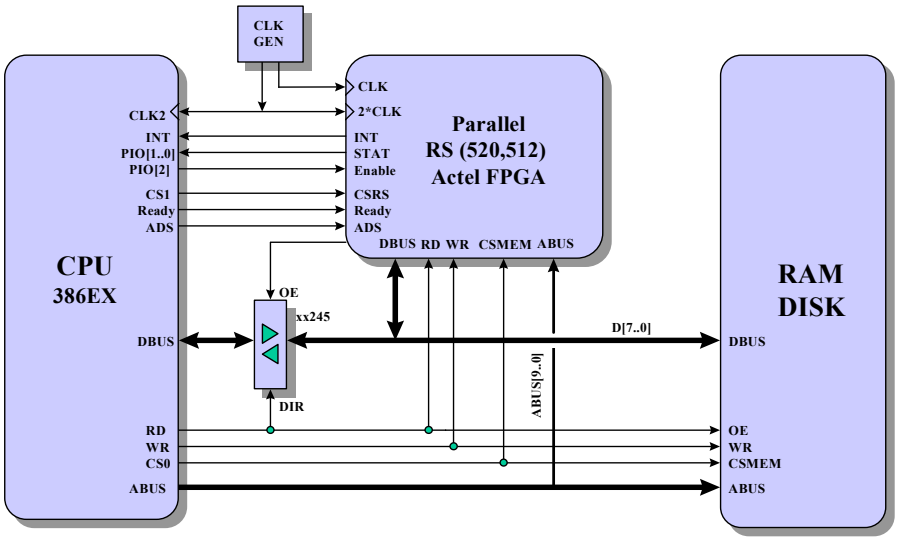


Figure 3 Example Codec Interface to a 386EX processor

3 Implementation of the RS Codec

The market in available devices was deemed to be unsatisfactory, being either too expensive, requiring glue-logic, or generally not meeting the requirements. Also there are no readily available codecs that would support data blocks of 512 bytes (the usual 8-bit RS code would support no more than 252 bytes) . We chose to implement our own device, based on FPGA technology, that being the least expensive and also available in space-qualified versions.

Choice of FPGA

The FPGA must cope with the space environment even if it is low cost. The low Earth orbit is however a relatively benign environment (for which the SEU statistics are in any case only available).

Our choice focused on the Actel series: the commercial A1020B and A1280A; and the radiation hardened RH1020 and RH1280. Both the commercial and RH version are immune to Single Event Latch up (SEL).

There is no significant difference in the susceptibility of the commercial and radiation hardened Codec to single event upsets (SEU). The difference between them lies in tolerance to total ionisation dose. Here the space hardened version is significantly better. In the LEO environment however the total radiation dose is not a problem.[5] There is no significant difference between commercial and space hardened in respect of linear energy transfer (LET) [6].

with option 1; and so is the 1280 using only its C modules. To implement option 2 we can only use the larger 1280 and need to use both its C and S modules.

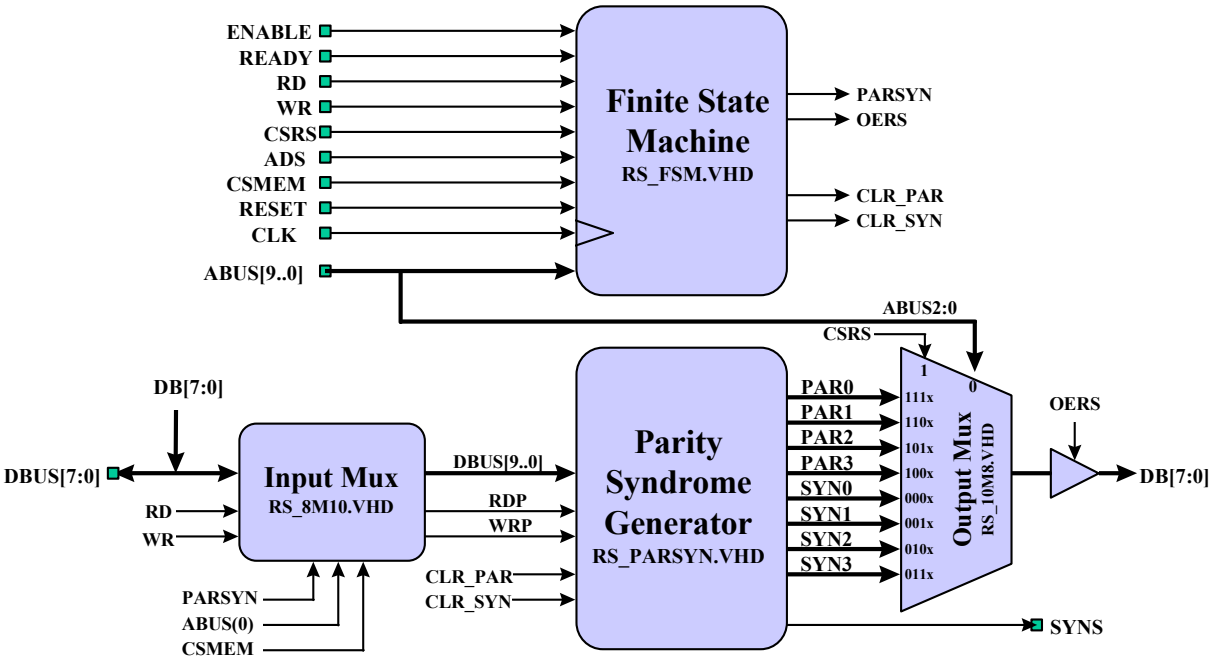
The LET threshold is significantly less for the S modules compared to the C modules. In the event of an SEU occurring in some specific S modules there is an expected failure mode in operation. We are working to replace these operationally critical S modules by C modules.

Structure of CODEC

Figure 3 shows a glue-less interface between an Intel 386EX micro controller, the Codec and the Ramdisk memory. The RS Codec "sits" on top of the data-bus between the CPU and the Ramdisk. An additional tri-state bi-directional line-driver is required to isolate the CPU from the Ramdisk data-bus when the RS codec pushes the parity or syndrome symbols onto the data-bus. This line-driver can be incorporated into the FPGA, however, this will impose a larger delay than obtained with an external line-driver. Also as shown in later figures the RS codec just fitted into a single 8000gates 1280 FPGA.

Option 1 system (fig4)

The design is divided up into four separate blocks, the Parity and Syndrome generator (RS_PARSYN), control block (RS_FSM) and the input and output multi-plexers



We have found that the 1020 is of sufficient size to cope

Figure 4 Option 1 architectural diagram

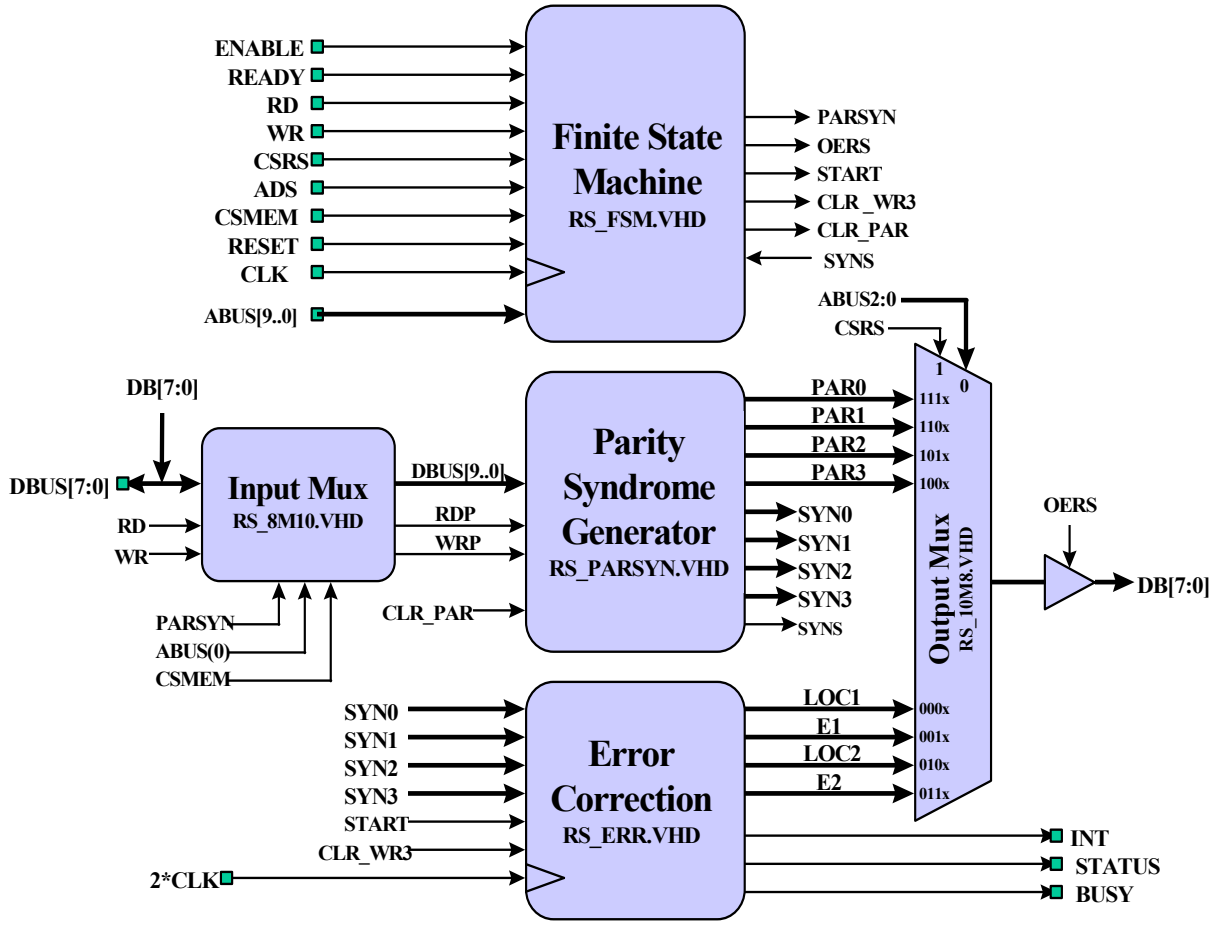


Figure 5 option 2 architectural diagram

(RS_8M10 and RS_10M8 respectively). The two multi-plexers translate the external 8 bits data bytes to internal used 10 bit words. The output multi-plexer reformats the 10 bits parity symbol in two 8 bits values. During a read operation of the parity bytes the input multi-plexer recombines these two memory locations back into a 10 bits word before passing it one to the Parity/Syndrome generator. For the data bytes the input multi-plexer simply adds the two most significant zeros.

Option 2 system(fig 5)

In addition to the option 1 blocks an Error Detection Block (RS_ERR.VHD) is added. This block receives the calculated syndrome symbols and tries to determine the error magnitude(s) and error location(s). By latching the four 10 bits syndrome symbols, the next information block can be processed whilst the Error Detection Block tries to determine the error location and magnitude. Further reduction in module count can be achieved by removing these latches. This will mean however that the driver software must wait if an error is detected before reading the next block.

4 Operation of CODEC

Encoding

The data is *encoded* using a standard *n-k* encoder. Blocks of data are written one byte at a time to memory and simultaneously into the Codec. After the last (512th) byte has been written to memory the 8 additional code bytes - expressing the 4 check words - are immediately available and are written to memory.

The software starts by writing the information bytes into memory. During this time the RS unit updates the parity bytes after each received information byte. After 512 information bytes are written to memory the driver software needs to write an additional 8 dummy write cycles. During this time the RS unit will disable the CPU from the data bus and push the parity symbols onto the data bus.

Decoding

Option 1

Stage 1 decoding is performed in hardware. Data is read from a block byte-by-byte, simultaneously into the CPU and into the Codec. Syndromes are immediately available after the last data byte has been processed. If the syndrome values are non-zero this information is passed onto the CPU. During a read operation the software driver reads 512 information bytes from the Ramdisk into the destination buffer. In our particular application this buffer is held in the main code memory which is protected by a majority voting system. After reading 512 information bytes the software driver needs to read the stored parity symbols. After the total of 520 read cycles the syndrome bytes are available immediately on completion of reading the block.

If the syndrome bytes are not all zero then the SYNS signal is asserted. The software driver must then try to compute error patterns e_i and e_j, and the error locations i and j (in the range 0 to 515). The computation may *fail* at this point indicating an extended error. If the calculation is successful then the driver must correct the erroneous bytes in the buffer memory.

Option 2

The read operation is similar to option 1. Two alternating receive buffers are required because the Error Detection block is processing the syndromes symbols in parallel with the Parity/Syndrome unit. The driver software reads a block of 520 symbols into the alternating buffer. If after reading 520 symbols the syndrome symbols are not all equal to zero than the syndrome symbols are passed to the Error Detection block. Whilst the next information block is being read the Error Detection unit determines the Error location and magnitude. After determining these values the CPU is interrupted. The driver software must then read the location and magnitude symbols and the correct the symbols in the previous buffer. This buffer can then be passed onto higher layers. For the last information block the driver needs to read one extra block in case the syndrome symbols are not all equal to 0. For this purpose the CPU can sample the Busy signal.

Design methodology

Our implementation of the Codec uses a single Actel FPGA (Field Programmable Gate Array) for the encoding and decoding process. This single FPGA implementation is designed by technology-independent VHDL and synthesised by Actel's Actmap synthesis tool. A software version of the RS code was first written in C and then translated into VHDL. This software version was also used to generate the test vectors for the hardware verification.

After running the design through the Designer Place and Route the software resulted in a total module usage of 84% of an 2000 gates Actel 1020 and only 23% of an 8000 gates 1280 FPGA.

5 Verification of the RS Codec

Option 1 and 2 were implemented in VHDL. For this the existing C-code was translated to structural VHDL code, synthesised to hardware using ACTEL's ACTMAP 3.1 and then run through the place and route software to obtain the module count usage (table 2). Figure 6 shows the design flow (without feedback) from concept to generating the FUSE file for the FPGA programmer.

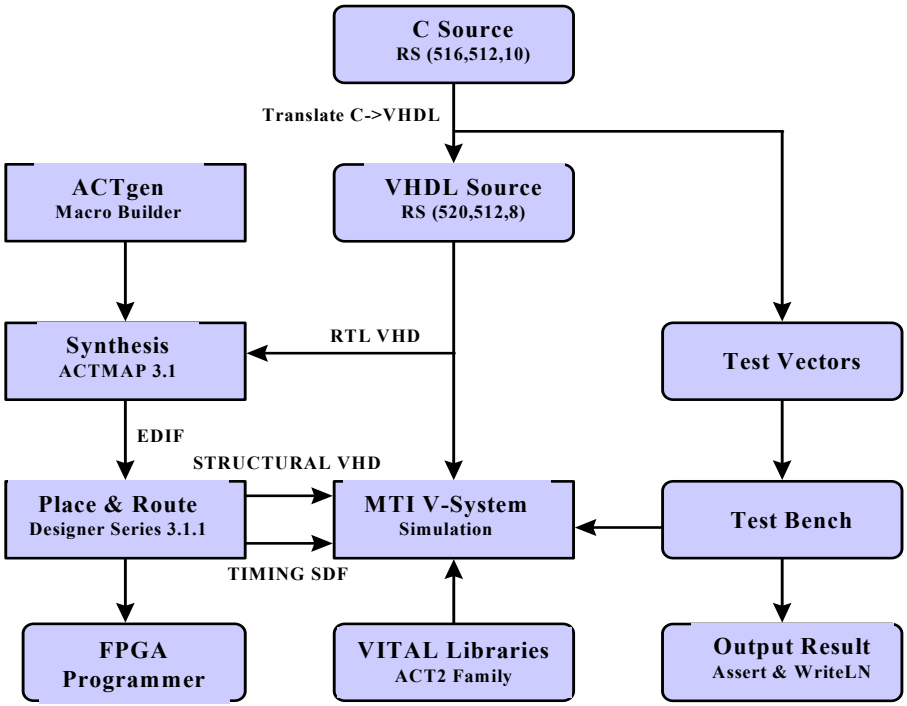


Figure 6 Codec Design and validation Flow

The VHDL code consisted of a hierarchical top file with small instantiated blocks such as finite state machines, combinatorial modules and multi-plexers (see figs 4,5). The design was divided into smaller logical blocks to improve modularity and to allow smaller blocks to be tested individually. Functional test benches were developed for the Parity/Syndrome generator block and Error detection block. After verification of these blocks they were instantiated into a top structural file and an overall test bench was developed.

For the functional verification the Codec design was instantiated together with a tri-state line driver and an SRAM model into the top test bench. The tri-state and SRAM model were downloaded from the web. The original C-program was used to generate random blocks of data with known error location and patterns. The data blocks and error information (location + pattern) were written to two separate files. The data block file was read by the test bench and the values from this file were asserted at appropriate times onto the Codec's data bus. The error file was used to verify the Codec-generated error location and pattern against the C-program generated one. The simulation results were displayed on the user screen using the VHDL *Assert* and *Writeln* constructs. The simulation stopped if the RS Codec calculated error location and/or magnitude values differed from the C program generated one. To obtain accurate timing information from the FPGA, VITAL libraries were used, with back-annotated values generated by the Designer Series.

ACTMAP	option	most speed? (FO=10)	worst delay (ns)	least area (FO=16)	worst delay (ns)
1020	1	504(92%)	141.4	462(84%)	93.5
1280 (C+S mod.)	1	308(25%)	41.2	282(23%)	44.4
1280 (C mod.)	1	484(80%	41.2	385(63%)	44.4
1280(C+S mod)	2	1232(100%)	134.7	1216(98.7%)	146.2

Table 2 Option 2 ACTMAP Synthesis results, Designer series 3.1.1

6 Summary of performance

The two options as described have yet to be implemented in hardware. The results presented in Table2 are from simulation only. They are believed to be a reliable indicator of future performance.

We show the synthesis results and Designer series with reported timing for option1 and 2 respectively. Either chip is suitable for option 1. Only the larger chip can be used for option 2 for which an 8000-gate 1280 FPGA was just sufficient. In order for this design to fit both the C and S modules needed to be utilised with a risk of lower reliability.

7.Conclusion

The paper has identified an application-specific low-complexity codec: the design (i) allows a natural (power of 2) block size of data; (ii) has DEC and EED capability; (ii) achieves a low complexity by implementing a non-standard method of decoding. The complexity of the completed code is sufficiently low that a single low-cost 8000 gates Actel 1280 FPGA is all that is needed for its implementation. We may note that buffering of the data is needed outside the Codec for full implementation.

References

[1] C. Paar, M. Rosner, **"**Comparison of Arithmetic Architectures for Reed Solomon Decoders" *FCCM 1997*, Napa valley, California, April 16-18, 1997.

[2] S. Lin and D.J Costello, "Error control coding: fundamentals and applications", Prentice Hall 1983

[3] S.B.Wicker V.J.Bhargava "Reed Solomon Codes and their applications" 1994 IEEE press

[4] C.I. Underwood, R. Ecoffet, "Observations of Single-Event Upset and Multiple-Bit Upset in Non-Hardened High-Density SRAMs in the TOPEX/Poseidon Orbit", *IEEE/NSREC Conference*, Snowbird, Utah, USA, Jul, 1993

[5] C.S. Dyer, A.J. Sims, P.R. Truscott, J. Farren, C. Underwood, "The Low Earth Orbit radiation Environment & its Evolution from Measurements Using the Cream & Credo Experiments".

[6] J.J.Wang, "Radiation Performance of ACTEL Products", Technology Development, Actel Corporation, Rev 1 October 1997