VOCAL

Reed Solomon Coding

The VOCAL implementation of Reed Solomon (RS) Forward Error Correction (FEC) algorithms is available in several forms. The forms include pure software and software with varying levels of hardware acceleration utilizing UDI or other custom hardware instructions. Pure software solutions are available in several different forms for both the encoder and decoder. Different versions allow speed versus memory tradeoffs to be made, and allow efficient and easy expansion of the code for assembly language optimization.

The Reed Solomon algorithms rely on special properties of finite-arithmetic Galois Field (GF) operations. The use of hardware acceleration for these operations can be used to greatly improve performance; for example, on the MIPS architecture, UDI/CorExtend instructions may be used for this purpose. Multiple levels of hardware acceleration are available, including single cycle multiply and inverse, as well as parallel multiplication and general-purpose bit-slicing/composite field operations.

VOCAL's embedded software libraries include a complete range of ETSI / ITU / IEEE compliant algorithms, in addition to many other standard and proprietary algorithms. Our software is optimized for execution on ANSI C and leading DSP architectures (TI, ADI, AMD, ARM, CEVA, LSI Logic ZSP, and MIPS). These libraries are modular and can be executed as a single task under a variety of operating systems or standalone with its own microkernel.

Algorithm Description

Reed Solomon codes are error-correcting codes that have found wide-ranging applications throughout the fields of digital communication and storage. Some of which include:

- Storage Devices (hard disks, compact disks, DVD, barcodes, etc.)
- Wireless Communication (mobile phones, microwave links, etc.)
- Digital Television
- Broadband Modems (ADSL, xDSL, etc.)
- Deep Space and Satellite Communications Networks (CCSDS)

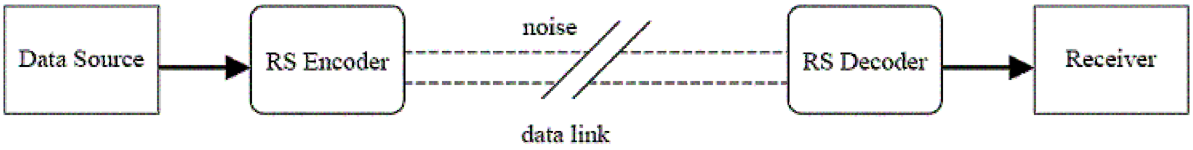


Figure 1. Application of Reed-Solomon Codes

RS codes are systematic linear block codes, residing in a subset of the BCH codes called non-binary BCH. It is block because the original message is split into fixed length blocks and each block is split into m bit symbols; linear because each m bit symbol is a valid symbol; and systematic because the transmitted information contains the original data with extra CRC or 'parity' bits appended.

These codes are specified as RS (n, k), with m bit symbols. This means that the encoder takes k data symbols of m bits each, appends n - k parity symbols, and produces a code word of n symbols (each of m bits).

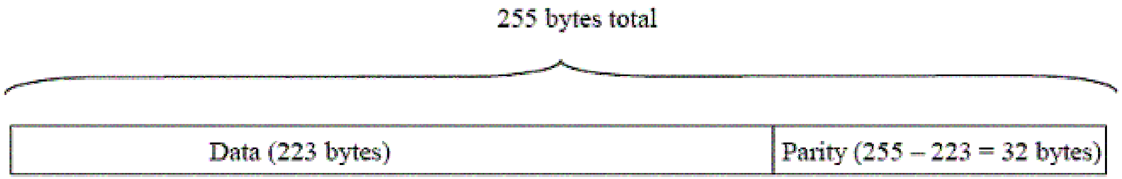


Figure 2. Breakdown of a RS(255, 223) Codeword

Reed Solomon codes are based on a specialized area of mathematics known as Galois fields (a.k.a. finite fields). These fields are of the form GF (p^m), where p is prime. RS makes use of Galois fields of the form GF (2^m), where elements of the field can be represented by m binary bits. Hence, RS codes of the form RS (2^8) lend themselves well to digital communication.

Primitive polynomials are of interest here because they are used to define the Galois field. A popular choice for a primitive polynomial is:

$$p(x) = x^8 + x^7 + x^2 + x^1 + 1$$

This is also known as the 0x87 polynomial, corresponding to the binary representation of the polynomial's coefficients excluding the MSB (i.e. 10000111). This specific polynomial is used in the CCSDS specification for a RS (255, 223). In GF (2^8) there are 16 possible primitive polynomials.

The VOCAL implementation has the ability to perform all combinations of RS (n, k) [n = 255, and 0 < k < n], for any of the 16 possible Galois fields, including the 0x87 field used by CCSDS. Additionally, the VOCAL RS modules can use any arbitrary generator polynomial for the calculation of the parity symbols.

Encoder

The Reed-Solomon encoder reads in k data symbols, computes the n - k parity symbols, and appends the parity symbols to the k data symbols for a total of n symbols. The encoder is essentially a 2t tap register where each register is m bits wide. The multiplier coefficients are the coefficients of the RS generator polynomial. The general idea is the construction of a polynomial; the coefficients produced will be symbols such that the generator polynomial will exactly divide the data/parity polynomial.

Decoder

The Reed-Solomon decoder tries to correct errors and/or erasures by calculating the syndromes for each codeword. Based upon the syndromes the decoder is able to determine the number of errors in the received block. If there are errors present, the decoder tries to find the locations of the errors using the Berlekamp-Massey algorithm by creating an error locator polynomial. The roots of this polynomial are found using the Chien search algorithm. Using Forney's algorithm, the symbol error values are found and corrected. For an RS (n, k) code where n - k = 2T, the decoder can correct up to T symbol errors in the code word. Given that errors may only be corrected in units of single symbols (typically 8 data bits), Reed-Solomon coders work best for correcting burst errors.

Reed Solomon Implementations

The implementations below can be customized to work with other RS (n, k) codes to yield similar results in performance.

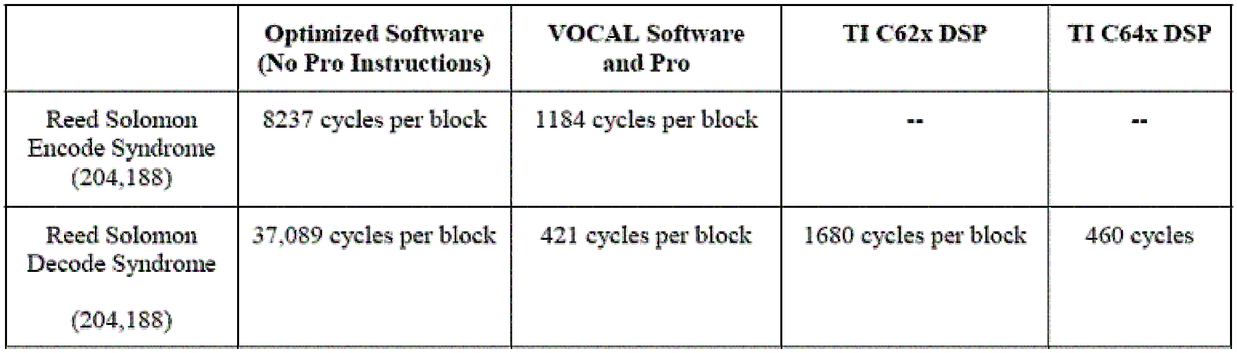
- Optimized Software Implementation. The pure software implementation is dominated computationally by multiplication over a finite field (Galois Field multiplication). The encoder requires 71,181 cycles per codeword on a MIPS32 processor and the decoder requires 66,045 cycles.

- Scalar GF Multiply Support. This is the simplest form of VOCAL's hardware acceleration. The Scalar GF Multiply Support extends the capabilities of the MIPS32 processor by taking advantage of MIPS Technologies CorExtend capability to decrease the number of cycles to 23,305 cycles to encode and 9,174 cycles per codeword to decode on the MIPS32 processor.

- SIMD GF Multiply Support. The SIMD GF Multiply Support requires 128 bytes of local ROM memory, but increases the performance to 3,918 cycles per megabit to encode and 3,078 cycles per codeword to decode.

- RS Encode Kernel. The RS Encode Kernel uses 1024 bytes of local ROM memory to encode. The number of cycles to process a codeword on a MIPS32 CPU falls to 2,702 cycles for encoding and decoding only consumes 828 cycles with this implementation.

Table 1 – Reed Solomon Benchmarks



	Optimized Software (No Pro Instructions)	VOCAL Software and Pro	TI C62x DSP	TI C64x DSP
Reed Solomon Encode Syndrome (204,188)	8237 cycles per block	1184 cycles per block	--	--
Reed Solomon Decode Syndrome (204,188)	37,089 cycles per block	421 cycles per block	1680 cycles per block	460 cycles

Reed Solomon Software Performance

The following table details performance numbers for a number of specific RS (n, k) implementations for two general purpose processing architectures and one digital signal processor. Numbers are provided for both decode in the presence of no error, as well as decode in the presence of maximum channel error. Note that correcting errors requires more processing power than simply validating blocks, and that the required processing power increases linearly with the error rate. Typical applications tend to keep the error rate low such that active correction is not required.

The two digit hexadecimal number in each column specifies the GF (255) primitive polynomial used to generate the underlying Galois field.

The listed performance numbers are:

- CPB - Cycles Per Block, how many CPU cycles are required to perform this step of the algorithm for each block of data
- MBPS @ 1GHz - maximum throughput in MBits/sec for each 1.0 GHz of processing power (bit rate measured on the data side, not the channel side.)

All measurements are for optimized C code for the particular architecture, compiled with GCC and -O4 optimizations. No hardware acceleration or SIMD instruction optimizations were used.

Table 2 – Reed Solomon X86 Performance

	RS (255,191) 0x1D		RS (255,223) CCSDS 0x87		RS (255,239) 0xCF		RS (255,247) 0x2D		RS (255,251) 0x63	
	CPB	**MBPS @ 1GHz**	**CPB**	**MBPS @ 1GHz**	**CPB**	**MBPS @ 1GHz**	**CPB**	**MBPS @ 1GHz**	**CPB**	**MBPS @ 1GHz**
Encode			10854	164.4	10552.5	181.2	6406.88	308.4	5075.25	395.6
Decode (no errors)			38994	45.8	19597.5	97.6	9045	218.5	4422	454.1
Decode (max errors)			120667	14.8	52511.25	36.4	20602.5	95.9	10251	195.9
Decode (max erasures)										

Table 3 – Reed Solomon ARM LINUX Performance

Platform	RS (255,191) 0x1D		RS (255,223) CCSDS 0x87		RS (255,239) 0xCF		RS (255,247) 0x2D		RS (255,251) 0x63	
	CPB	**MBPS @ 1GHz**	**CPB**	**MBPS @ 1GHz**	**CPB**	**MBPS @ 1GHz**	**CPB**	**MBPS @ 1GHz**	**CPB**	**MBPS @ 1GHz**
Encode	60000	25.47	25500	69.96	20250	94.42	14750	133.97	11250	178.49
Decode (no errors)	57500	26.57	31750	56.19	18375	104.05	5166.67	382.45	2583.33	777.29
Decode (max errors)	449500	3.4	199875	8.93	90875	21.04	39833.33	49.61	17875	112.34
Decode (max erasures)	760000	2.01	345250	5.17	153500	12.46	80250	24.62	34041.67	58.99

Table 4 – Reed Solomon ARM LINUX Performance

Platform	RS (255,191) 0x1D		RS (255,223) CCSDS 0x87		RS (255,239) 0xCF		RS (255,247) 0x2D		RS (255,251) 0x63	
	CPB	MBPS @ 1GHz	CPB	MBPS @ 1GHz	CPB	MBPS @ 1GHz	CPB	MBPS @ 1GHz	CPB	MBPS @ 1GHz
Encode	190000	8.04	110000	16.22	65000	29.42	35000	56.46	20000	100.4
Decode (no errors)	195000	7.84	100000	17.84	45000	42.49	25000	79.04	133333.33	150.6
Decode (max errors)	525000	2.91	240000	7.43	110000	17.38	55000	35.93	30000	66.93
Decode (max erasures)	680000	2.25	325000	5.49	155000	12.34	80000	24.7	40000	50.2