Product Specification

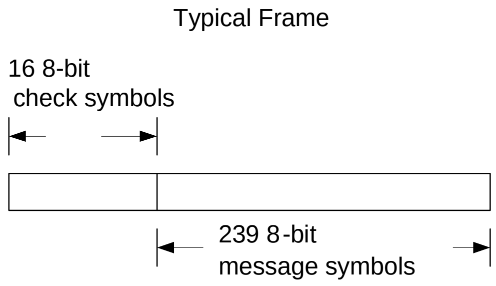
Salamander

SALyy222D

Generic high speed (255,239) Reed-Solomon Decoder

Features

- n = 255, k = 239, m = 8-bit symbols
- Capable of correcting up to (16) 8-bit symbol erasures
- Support of shortened frames
- Support of punctured codes
- Errors and erasures
- High-speed, low latency
- Uses the Berlekamp-Massey algorithm
- Simple handshake protocol for reliable interfacing
- Fully synchronous design
- Low speed / low power operation
- Comprehensive verification plan provided



General Description

The SALyy222D consists of verilog IP for implementing a 16 erasure correcting Reed Solomon forward error correction decoder.

Incoming data are treated as coefficients of a polynomial over the field GF(256) generated by the polynomial

$$F(x) = x^8 + x^7 + x^2 + x + 1 \tag{1}$$

The device uses the known roots of the generator polynomial to construct a set of syndromes which result when the code word is evaluated at each of its 2t roots. Those syndromes are then used to derive a polynomial whose roots are the locations of all the detected errors in the received word. The locations of the errors are then determined using the Chien root locator and the magnitudes of the errors are found using the Forney algorithm. The error locations and magnitudes are then used to fix the symbols found to be in error.

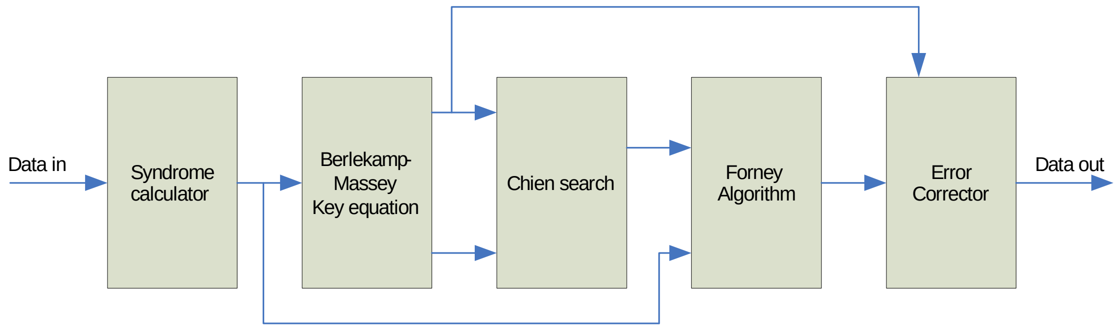


Figure 1: Decoder Block Diagram

Theory of Operation

What are Reed-Solomon codes?

There are a large number of excellent references available on the theory and design of Reed-Solomon codes. We'll provide only a very concise overview:

Perhaps the easiest way to understand RS codes is to look at them the way they were originally formulated by Reed and Solomon, namely, as polynomial codes over finite fields.

First of all, what is a "finite field"? Without going into the gory details, a finite field is simply a finite set of numbers (always having a prime number or a power of a prime number of elements) over which the familiar operations of addition, subtraction, multiplication and division are well defined. Additionally, we require the operations on the finite set to exhibit the commutative, associative, and distributive properties and to contain identity and reciprocal elements. The important things about finite fields from the standpoint of coding theory are: 1) that they are closed, i.e., addition or multiplication of two field elements always results in an element of the field, and 2) that successive powers of certain field elements (called primitive elements) will always result in unique elements of the field, until all the elements of the field are used up, thus "generating" the field.

The field we are interested in is called GF(256), i.e., 2^8. This field can be represented conveniently in 8-bit quantities (bytes).

Now, denote the message that you are trying to encode as a sequence of k bytes. Think of the bytes as elements of the finite field GF(256), and think of the bytes as coefficients of a message polynomial:

$$m(x) = m_0 + m_1 x + m_2 x^2 + \ldots + m_{n-1} x^{n-1} \quad (4)$$

Now, a Reed-Solomon code word **c** is formed by evaluating the message polynomial at each of the elements of the finite field. Thus, taking λ to be a primitive element:

$$\mathbf{c} = [m(0), m(\lambda), m(\lambda^2), \ldots , m(\lambda^{n-1})] \qquad (5)$$

Though easy to understand, this method of code construction has fallen out of favor in favor of the generator polynomial approach, which is much easier to implement and decode in hardware. This approach takes advantage of the fact that RS codes are linear and cyclic, meaning that the sum of any two code words is always a code word, and a cyclic shift of any code word always results in a code word. Thus a code word c(x) can be formed by multiplying the message polynomial by a special polynomial called a generator polynomial, which has as its roots 2t consecutive powers of λ.

$$c(x) = m(x)g(x) \qquad (6)$$

$$g(x) = (x - \lambda^0)\,(x - \lambda)\,(x - \lambda^2)\ldots\,(x - \lambda^{2t-1}) \qquad (7)$$

Finally, in practical applications, the code words are "systematic," meaning that the message appears intact in the code word followed by the "parity" symbols. This is achieved by dividing a shifted version of the message polynomial by the generator polynomial and adding the remainder to a shifted version of the message.

$$c(x) = \mathrm{rem}[x^{n-k}m(x)/g(x)] + x^{n-k}m(x) \quad (8)$$

Where rem[] is defined as the remainder polynomial resulting from the polynomial division operation over GF(256).

Signal Descriptions

The module pinout is shown in the figure below, and in table 1. The signals are conveniently organized into functional groups as follows:

Clock and Reset

The design is fully synchronous with a single clock signal. The reset signal is synchronous and needs to be asserted for at least one full clock cycle to reset internal logic. Because ***rst_n*** is designed to be connected to a system-wide synchronous reset, there is an additional soft reset signal, ***init***.

Control signals

Two signals control flow of data into the device, ***din_rdyin_n*** and ***din_rdyout_n***. The ***din_rdyout_n*** signal indicates that the device is ready to receive data. The ***din_rdyin_n*** signal indicates that data into the device is valid. Valid data is being shifted into the device when both ***din_rdyin_n*** and ***din_rdy_out_n*** are asserted (low).

Two signals control flow of data out of the device, ***dout_rdyin_n*** and ***dout_rdyout_n***. The ***dout_rdyout_n*** signal indicates that data out of the device is valid. The ***dout_rdyin_n*** signal indicates to the device that it's OK to shift data out. Valid data is being shifted out of the device when both ***dout_rdyin_n*** and ***dout_rdy_out_n*** are asserted (low).

Data signals

The data are clocked in on ***din*** and clocked out on ***dout***. The ***erase*** signal is used to mark an erasure by asserting it while shifting data into the device. The data on ***din*** will be ignored for erased symbols.

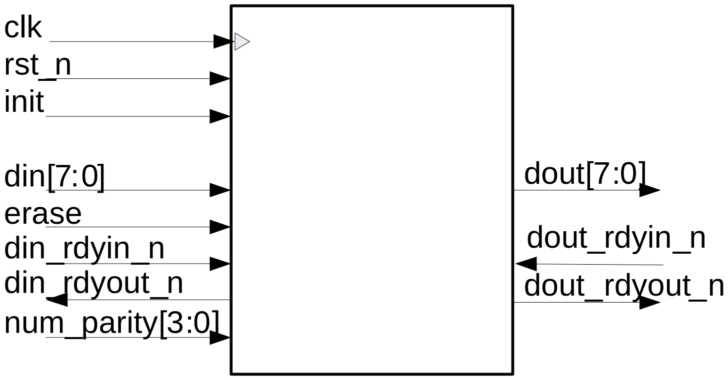


Figure 2: Component pinout

Pin	Sense	Width	Description
clk	in	1	clock
rst_n	in	1	synchronous reset
init	in	1	device “soft” reset
din	in	8	serial data (code word symbol) in
erase	in	1	mark the current symbol as “erased”
din_rdyin_n	in	1	data in is valid
din_rdyout_n	out	1	device is ready to receive data
dout	out	8	data out
dout_rdyin_n	in	1	interface is ready to receive data from device
dout_rdyout_n	out	1	data out is valid
num_parity	in	4	number of parity symbols

Table 1. Component pinout

Waveforms

Input

The input functional timing is shown below. ***Din_rdyin_n*** is used as an input data enable, ***din_rdyout_n*** is used to indicate when data (as opposed to parity) is being shifted into the device.

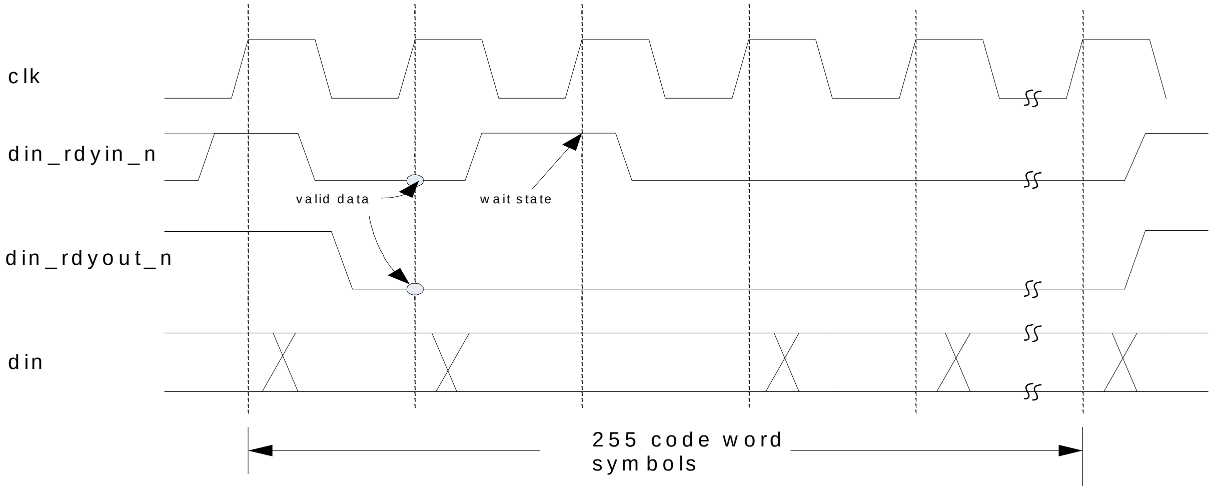


Figure 3: Input timing

Output

The output functional timing is shown below. ***Dout_rdyout_n*** is used as an output data ready indication, ***dout_rdyin_n*** is used to signal the device that it's OK to shift data out.

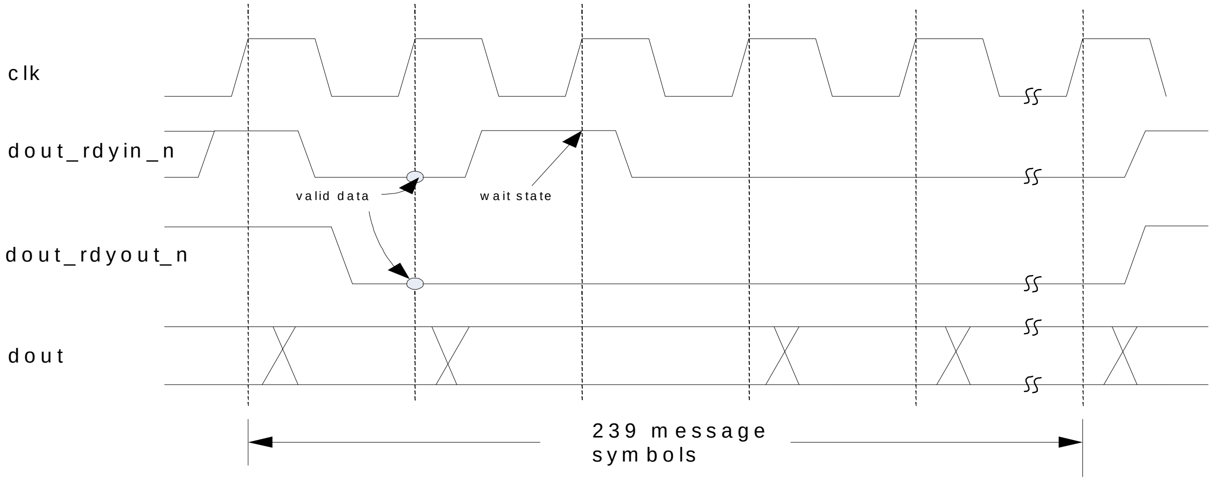


Figure 4. Output timing

Module Verification

The SALyy222D has been subjected to extensive verification to ensure the highest quality product possible. A comprehensive test plan was implemented which included the following:

- High-quality random data source
- High-quality random noise source
- Extensive flow-control simulations
- Verification of operation against known data sequences

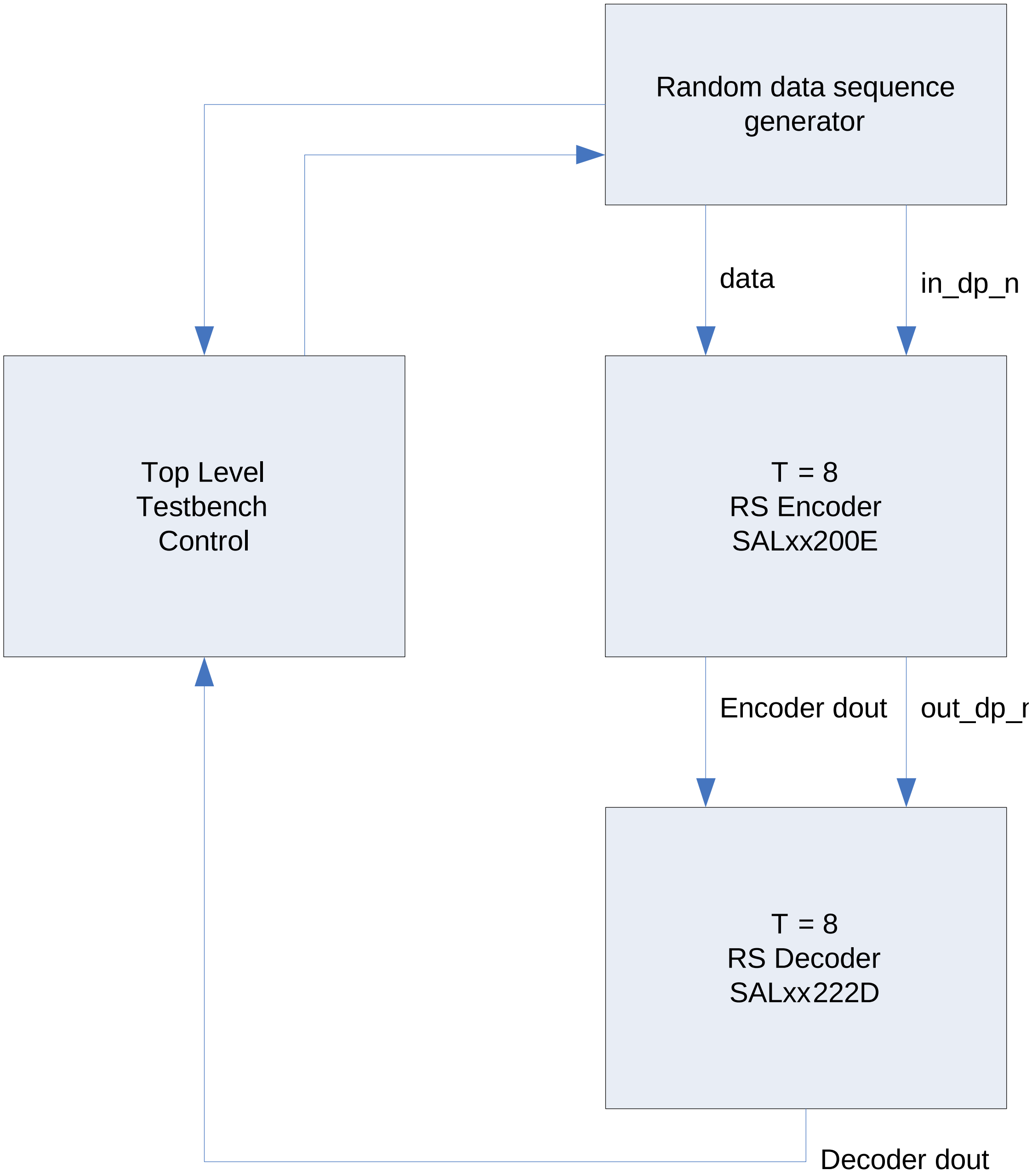
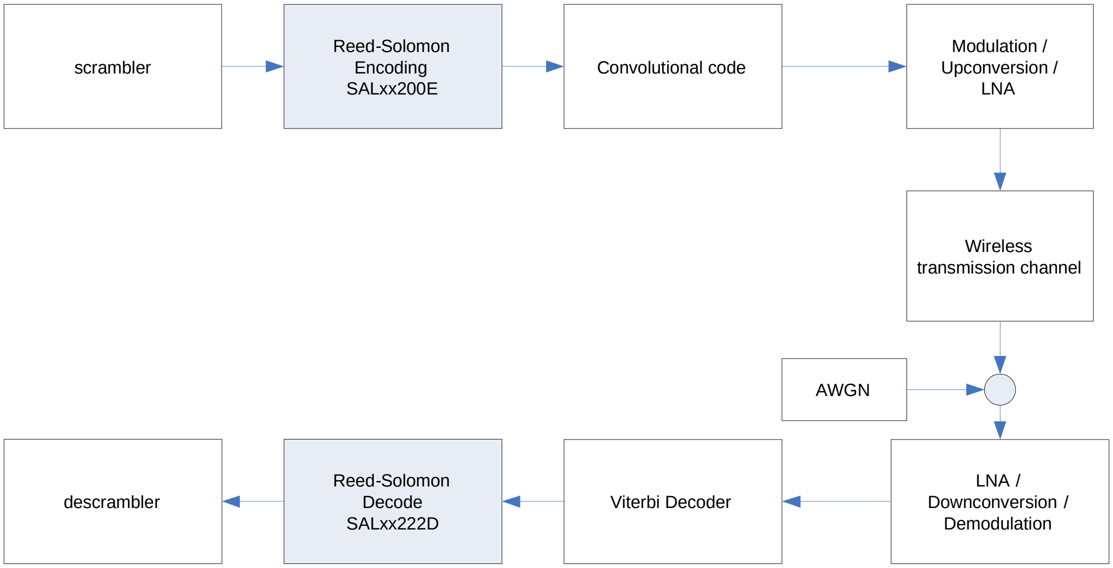


Figure 5: Testbench Block Diagram

Application: Wireless Internet System

The Reed-Solomon code forms an integral part of many wireless Internet telemetry systems.



Ordering Information

Salamander Error Correction currently has 16 generic Reed-Solomon decoder IP modules available, as well as a custom device design service:

SALyy222D (255, 239) poly = $x^8 + x^7 + x^2 + x + 1$, RS decoder, high speed, erasures only

About Salamander:

Salamander Error Correction develops and sells error correction modules of the highest quality worldwide.

Salamander Error Correction is a division of Komodo Industries, Inc.

Salamander Error Correction:
3364 Via Alicante
La Jolla, CA 92037

sales@salamander-ecc.com