



Features

- $n = 255, k = 223, m = 8$ -bit symbols
- Capable of correcting (16) 8-bit symbols
- Support for shortened frames
- Punctured frame support
- 2 clock cycle latency
- Simple handshake protocol for reliable interfacing
- Fully synchronous design
- Very high speed operation
- Comprehensive verification plan provided

General Description

The SALyy201E consists of verilog IP for implementing the (255, 223) 16-error-correcting Reed Soloman forward error correction encoder.

The device treats incoming data as coefficients of a message polynomial over the field GF(256) generated by the polynomial

$$F(x) = x^8 + x^7 + x^2 + x + 1 \quad (1)$$

Systematic code words are formed by dividing a shifted version of the incoming data (a message polynomial) by a so-called generator polynomial, taking the remainder as the “parity” portion of the code word:

$$c(x) = x^{2t}m(x)/g(x) + x^{2t}m(x) \quad (2)$$

Such a code is t -error-correcting, where $t = (d_{min}-1)/2$.

The generator polynomial is given in the specification as:

$$g(x) = (x-\lambda^0)(x-\lambda)(x-\lambda^2)\dots(x-\lambda^{2t-1}) \quad (3)$$

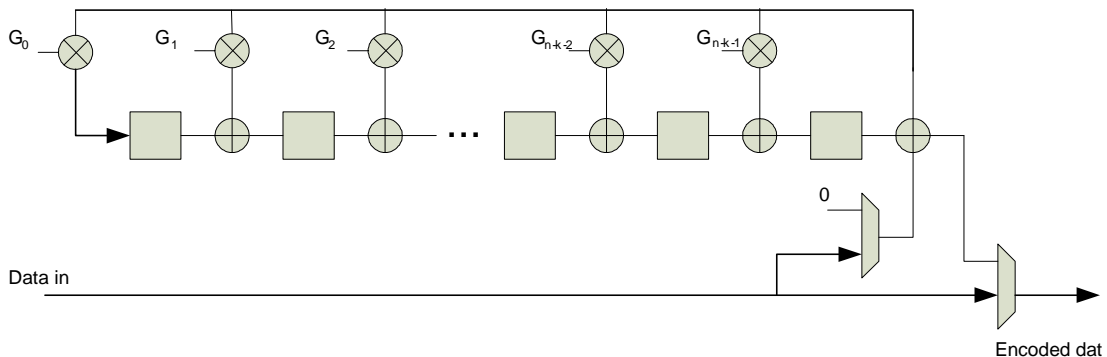
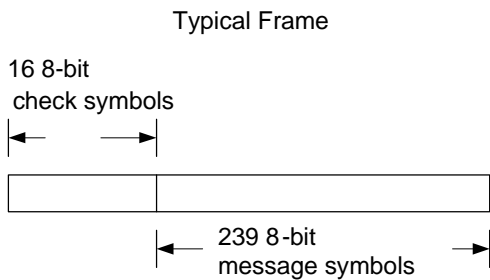


Figure 1: Encoder Block Diagram

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:

$$m(x) = m_0 + m_1x + m_2x^2 + \dots + m_{n-1}x^{n-1} \quad (4)$$

Now, a Reed-Solomon code word \mathbf{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:

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

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 .

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

$$g(x) = (x-\lambda^0)(x-\lambda)(x-\lambda^2)\dots(x-\lambda^{2t-1}) = 0 \quad (7)$$

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.

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

Where $\text{rem}[]$ 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.

Control signals

Two signals control flow of data into the device, *din_rdyin_n* and *in_dp_n*. The *din_rdyin_n* signal indicates that data into the device is valid. The *in_dp_n* signal indicates that message bits are being shifted into the device.

Two signals control flow of data out of the device, *dout_rdyout_n* and *out_dp_n*. The *dout_rdyout_n* signal indicates that data out of the device is valid. The *out_dp_n* signal indicates when message bits are being shifted into the device, as opposed to parity bits.

Data signals

The data are clocked in on *din* and clocked out on *dout*.

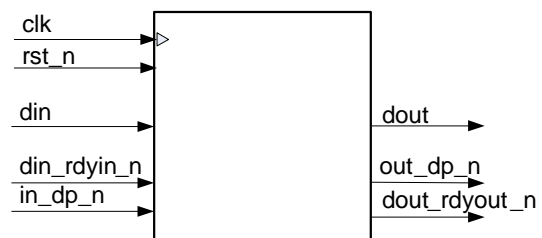


Figure 2: Component pinout

Pin	Sense	Width	Description
clk	in	1	Clock
rst_n	in	1	Synchronous reset
din	in	8	Serial data (message) in
din_rdyin_n	in	1	Indicates serial data in is valid
in_dp_n	in	1	'1' indicates data in, '0' = ignore input (during parity phase)
dout	out	8	Data out
out_dp_n	out	1	When '1', indicates data out, else it's parity
dout_rdyout_n	out	1	Indicates that data is available to be shifted out

Table 1. Component pinout

Waveforms

Input

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

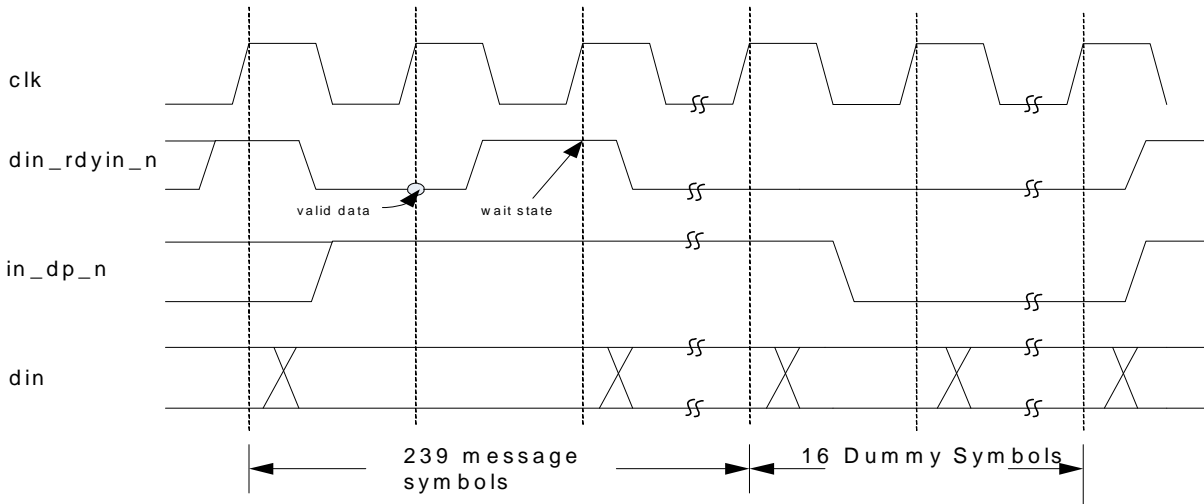


Figure 3: Input timing

Output

The output functional timing is shown below. *Dout_rdyout_n* is used as an output data ready indication, *out_dp_n* is used to indicate when data (as opposed to parity) is being shifted out of the device.

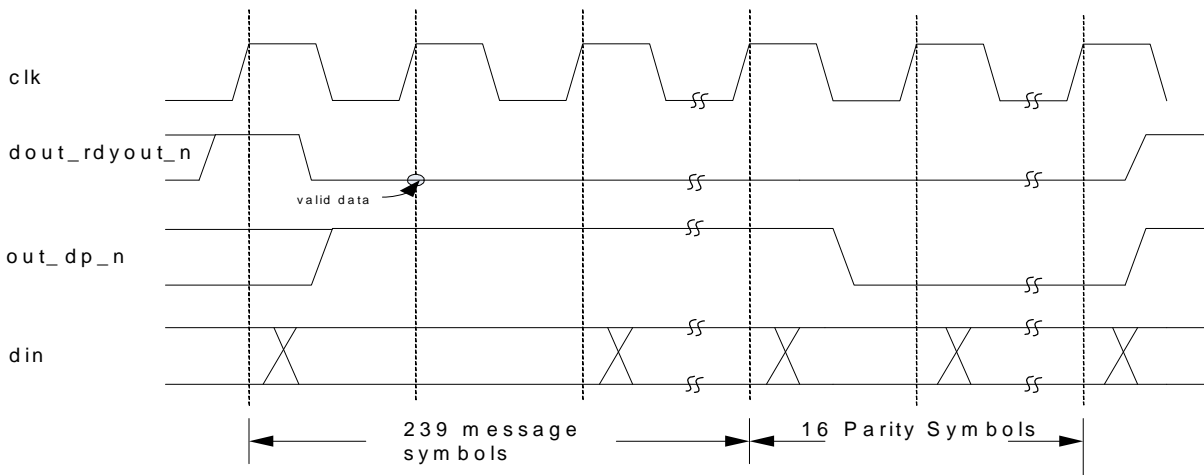


Figure 4. Output timing

Module Verification

The SALyy201E 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

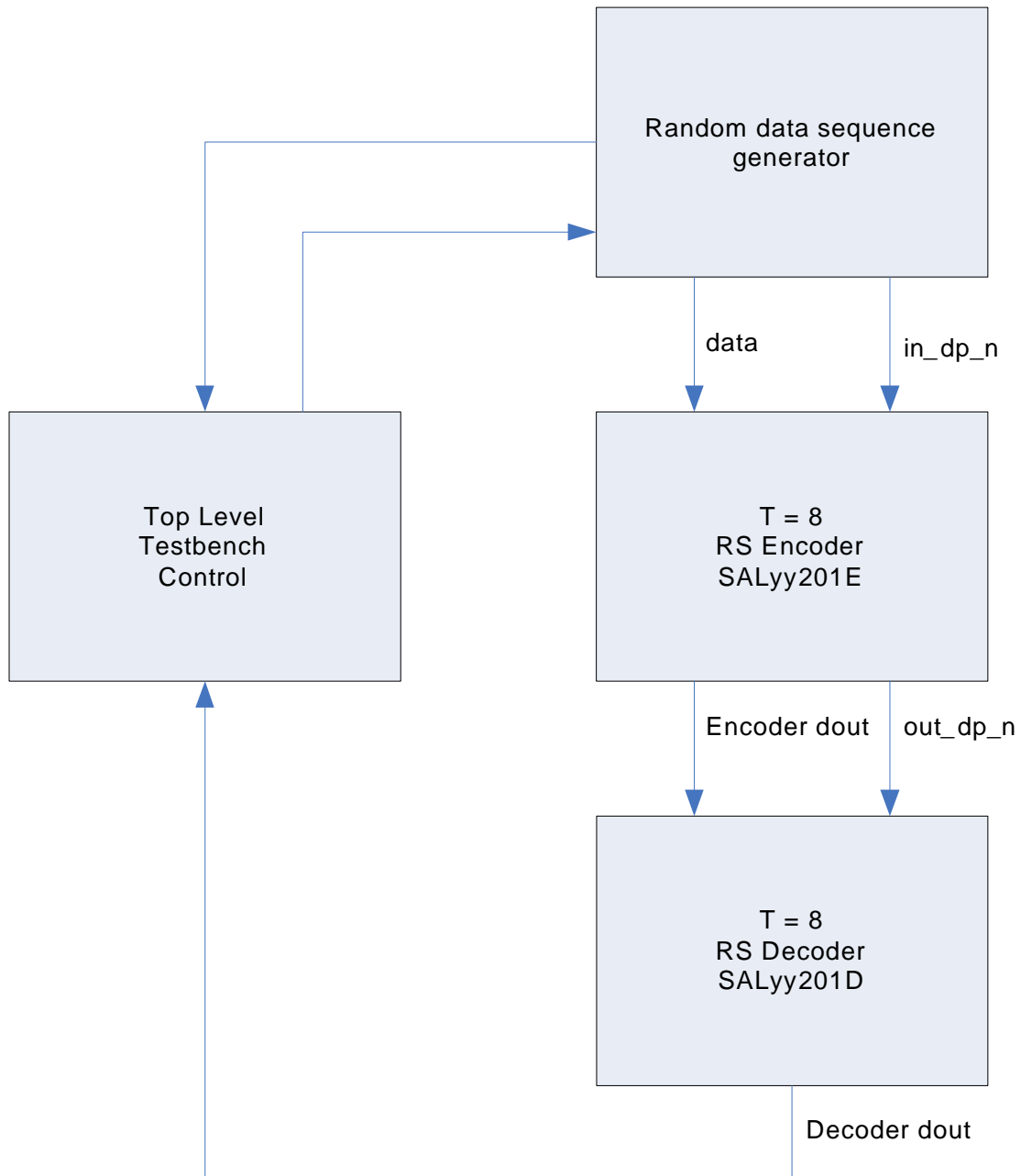
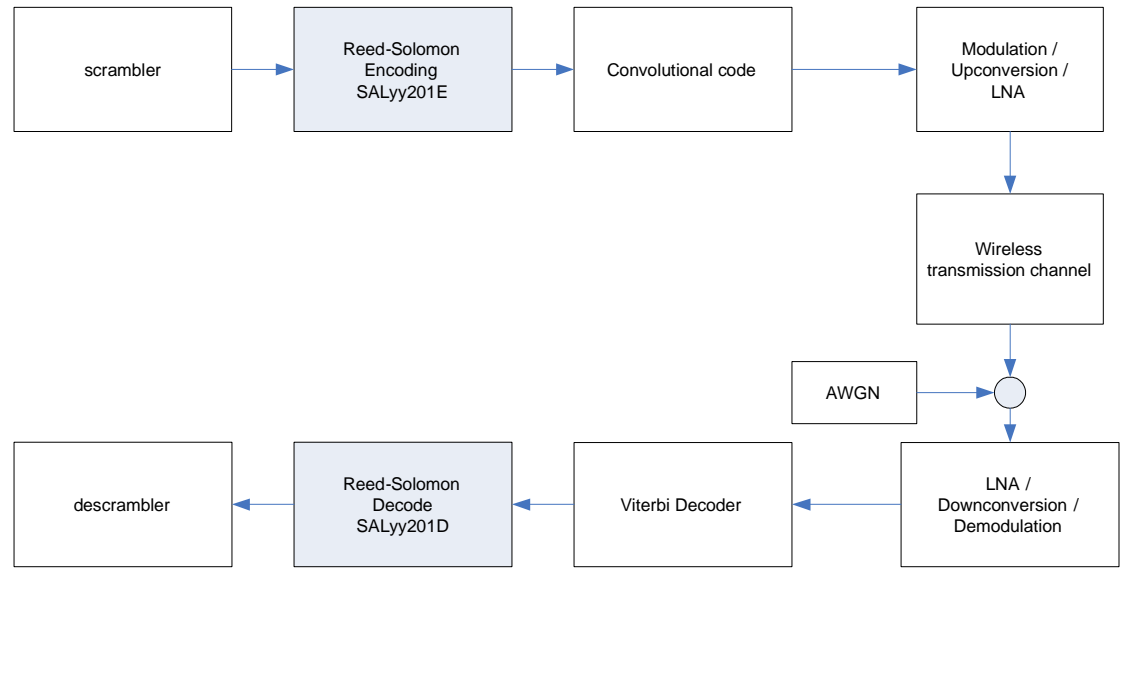


Figure 5: Testbench Block Diagram

Application: Metropolitan Area Network wireless Internet System

The Reed-Solomon code forms an integral part of a wireless internet communication system.



Ordering Information

Salamander Error Correction currently has 3 generic Reed-Solomon encoder IP modules available:

SALxx200E (255, 239) RS encoder, field polynomial = $x^8 + x^4 + x^3 + x^2 + 1$

SALyy200E (255, 239) RS encoder, field polynomial = $x^8 + x^7 + x^2 + x + 1$

SALyy201E (255, 223) RS encoder, field polynomial = $x^8 + x^7 + x^2 + x + 1$

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