RSTK
(Reed-Solomon Toolkit)

Author: John Willy
Date: May 17, 2003
Revision: 0.1
Last Rev: 0.0

Revision History Sheet

REV	DATE	Description	Initials
0.1	17-May-03	Initial document	JSW

Table of Contents

1 Description

RSTK is a C language program that generates Reed-Solomon HDL source code modules that can be compiled and synthesized. The supported languages are

- C
- Verilog
- VHDL

Although ‘C’ Reed-Solomon files can be generated the code is primarily designed to emulate the hardware functions of a Reed-Solomon Encoder/Decoder. When compiled and executed the operation of an encoder or decoder can be verified. It can be used to check the Verilog or VHDL code is functioning correctly and if not, provide a tool analyze and debug the modules. It is not designed as an optimized software library routine.

RSTK produces the following core modules:

- RS-Multiply – Scalar Field and General Field Multipliers
- RS-Encoder
- RS Decoder Syndrome Calculation
- RS Berlkamp-Massey Error Locator Polynomial
- RS Chien Search – Polynomial Roots (Error Location)
- RS Forney – Correction Value (Error Magnitude)
- ROM Lookup tables for the Decoder for Xilinx or Altera FPGAs
- Simulation support files for VHDL and Verilog simulators

RSTK does not generate a complete RS codec, it generates the core modules of an RS Encoder/Decoder. These modules can be modified as necessary to create an Encoder/Decoder for a specific design. Each module is designed as functional block. These blocks can be used independently of another design or combined to create a full Reed-Solomon Encoder/Decoder.

2 RS Polynomials

The following predefined primitive polynomials are options that are selectable from the command line.

ITU381/INTELSAT:
GF(2^8)
P(x) = x^8 + x^7 + x^2 + x + 1 = $110000111_{2} = 187_{16} = 391_{10}$

DVB/CD/ATSC:
GF(2^8)
P(x) = x^8 + x^4 + x^3 + x + 1 = $100011101_{2} = 11d_{16} = 285_{10}$

X4:
GF(2^4)
P(x) = x^4 + x^1 + 1 = $10011_2 = 13_{16} = 19_{10}$
X5:
GF(2^5)
P(x) = x^5 + x^2 + 1 = $100101_2 = 25_{16} = 37_{10}$

User defined polynomials for GF(2^n) fields can be entered as the binary representation of the polynomial i.e. 10011 entered a –p10011 will produce the same code as using the X4 selection.

3 Command Line

RSTK is executed using the command line from the Unix shell or DOS command window (NT/Win2K)

Example:

```
rstk -pitu -n8 -c3 -b120 -gvhdl
```

Command Line Options:

-p Galois field generator polynomial

–pitu, -pdvb, -p10101010, -px4, -px5

itu polynomial GF(256) :
dvb polynomial GF(256) :
x4 polynomial GF(16) :
x5 polynomial GF(32) :

-n number of bits in codeword symbol.

This represents the Galois Field in binary as 2^n. A codeword with 8 bits is represented as GF(2^8) or GF(256)

-c Number of codeword symbols to correct

This is the number of symbols the RS codec will correct in a code block. For example a code block is usually described as rs(255,251). 255-251 = 4. The number of codeword symbols that can be corrected is 4/2 or 2. For this code c=2.

-b Code generator base

This is the base of the generator polynomial. The generator polynomial is defined as $(x+\alpha^{base})(x+\alpha^{base+1})\ldots(x+\alpha^{base+2*c-1})$. Where c is the number of symbols to correct in a code block. If the base is greater than 1 an additional log multiplier ROM lookup table will be produced to normalize the error correction values produced by the Forney algorithm to the base of the generator polynomial.

-g Code to create:
C – C source code

vlog - Verilog source code
vhdl – VHDL source code

-r Simulation support files:

sim - Verilog & Vhdl simulation support files (*.hex)

4 Compiling RSTK

4.1 RSTK Source Files

argtable.c
gen_gf_tables.c
gen_rs_berl.c
gen_rs_chien.c
gen_rs_enc.c
gen_rs_forney.c
gen_rs_mul.c
gen_rs_rom.c
gen_rs_syndec.c
gf_arith.c
poly_math.c
rs_c_util.c
rs_tk.c

argtable.h
rsccg.h
rsccg_proto.h

4.2 RSTK Program Modules

void rsgen_gf_tables(unsigned char *, int, int, int) ;

Generate Galois Field tables from field generator polynomial.
The tables produced are:
gf_alpha_tbl[] – Array of GF(2^n) field elements a^n
gf_invalpha_tbl[] – Array of 1/(a^n)
gf_powalpha_tbl []– Array of (a^n)^n

unsigned char factor_poly(unsigned char *, int, int, int) ;

Divides each field element mod2 of GF(2^n) by the primitive polynomial and returns alpha (a^n), the remainder of the division by P(x)
For GF(256) the field element a^8 is:

(x^8+0+0+0+0+0+0+0)/(x^8 + x^7 + x^2 + x + 1) =
100000000/110000111 = 010000111

This routine creates the table of a^n for n=0 to n=2^n. This table is used by all other code modules to generate the ROM lookup tables and scalar multipliers.

unsigned char gf_mul(unsigned char , unsigned char, int gf_poly) ;

Multiplies two field elements and returns result.

unsigned char gf_add(unsigned char, unsigned char) ;

Adds two field elements and returns result.

int gf_poly_mul(unsigned int *, int, unsigned int *, int, int) ;

Multiplies two polynomials and returns the degree of the result.

void gen_rs_scalar_mul(FILE *stream, unsigned int, unsigned int, unsigned int) ;

Generates the code for scalar multipliers of the RS encoder and RS syndrome decoder.

void gen_rs_mul(FILE *stream, int, unsigned int) ;
Generates the code for the general field multiplier for GF(2^n).

void gen_rs_rom(FILE *stream, unsigned char * , unsigned char *, int , int);
Generates the ROM lookup tables for Altera and Xilinx FPGAs. Also generates *.hex initialization files for simulation of ROM lookup.

int expand_gen_poly(int , int , int) ;

Expands the generator polynomial $(a^{base}-1)(a^{base+1}-1)\ldots(a^{base+2*c-1}-1)$
The coefficients of the expanded generator polynomial are used to generate the Scalar multipliers.

void gen_gfarith_hdr(FILE *stream, int) ;
Creates the library package header for the VHDL gf_arith.vhd file

void gen_scalar_proto(FILE *stream, unsigned int , unsigned int , unsigned int) ;
Creates the library package scalar multiply function prototypes for the VHDL gf_arith.vhd file

void gen_multiply_proto(FILE *stream, unsigned int ,unsigned int) ;
Creates the library package general multiply function prototype for the VHDL gf_arith.vhd file

void gen_proto_footer(FILE *stream, unsigned int) ;
Creates the library package trailer for the VHDL gf_arith.vhd file

void gen_gfarith_footer(FILE *stream, unsigned int) ;
Creates the package body trailer for the VHDL gf_arith.vhd file

void gen_rom_init(FILE *stream, unsigned char *, unsigned char *, int , int) ;
Creates the ROM files for simulation or synthesis

void gen_rs_encoder(FILE *stream, unsigned int *, int, int , int) ;
Generates RS Encoder

void gen_rs_syndec(FILE *stream, int, int, int , int) ;
Generates RS RX Syndrome Decoder

void gen_rs_berl(FILE *stream, int, int, int , int, int) ;
Generates RS Berlkamp-Massey Error Locator.

void gen_rs_polyreg(FILE *stream, int, int, int , int, int) ;
Generates polynomial shift registers for the Berlkamp-Massey Error locator.

void gen_rs_chien(FILE *stream, int, int, int , int, int) ;
Generates RS Chien Search Error Location.

void gen_rs_forney(FILE *stream, int, int, int , int, int) ;
Generates RS Forney Error Correction.

void gen_rs_cutil(FILE *stream) ;

int vector_len(int) ;

void conv2_bvec_str(int, int, char *) ;

void pr_vec_str(FILE *stream, int , char *) ;

4.3 UNIX/Linux

RSTK is a console application under Unix/Linux systems. Add the RSTK files listed above to the project directory. The make file below can be used to build the application.

```
================ Makefile ================
CCFLAGS = -g -Wall
LDFLAGS = -g
CMD=rstk

.c.o:
	cc $(CCFLAGS) -c $<

$(CMD):	argtable.o gen_gf_tables.o gen_rom_tables.o gen_rs_berl.o
gen_rs_chien.o gen_rs_enc.o gen_rs_forney.o gen_rs_mul.o gen_rs_rom.o
gen_rs_syndec.o gf_arith.o poly_math.o rs_c_util.o rs_tk.o
	cc $(LDFLAGS) $^ -o $@


test:	$(CMD)
	./$(CMD) -pitu -n8 -c3 -b120 -gC >testlog 2>&1

clean:
	rm -f $(CMD) *.o makelog testlog
================
```

4.4 Microsoft Visual C++

RSTK is a console application under MS Windows. Create a new project with MS Visual C/C++ as a Win32 console application. Add the files listed above to the project. Build the application. Open a DOS command shell to run the program.

5 HDL Code Generation

RSTK currently supports GF fields from 2^3 to 2^8,GF(8) to GF(256) for C/C++, Verilog and VHDL languages. The code produced for the Reed-Solomon encoder/decoder is designed to support shortened code blocks. For example, the encoder will zero the syndrome generator when start_enc is asserted and transmit only the number of code word symbols that are in the block. On receiving a code block the syndrome calculation is performed on the entire field size i.e. from 0 to 255 for a GF(256) code. This will result in the correct error location polynomial syndromes for the error location algorithm (BERL_MAS), which is equivalent to prepending zeros back into the code block. For example: A RS code of (94,88) GF(256) with the code block length of K=88 data bytes and 6 check bytes (which can correct 3 codeword symbols), the encoder parameters are (N,K) where N=2*t+K, and t is the number of codeword symbols to be corrected and K is the number of symbols in a code block.

6 VHDL/Verilog/C RS Core Modules

6.1 RS Encoder

Port Signal Name	Input/Output	Function
rst_n	In	Resets core to initial state
clk	In	Clock
start	In	Start encode
d_in	In[x..0]	Data Input Bus
enc_dout	Out[x..0]	Encoder Data Output
enc_dov	Out	Encoder data output valid
enc_synv	Out	Encoder Syndrome valid
enc_done	Out	Encoder done

‘C’ Example

```
extern unsigned char msg_in[] ;
extern unsigned char enc_msg_out[] ;

void rs_encode(int n, int k, char dbg)

rs_encode(15,13,0) ;
```

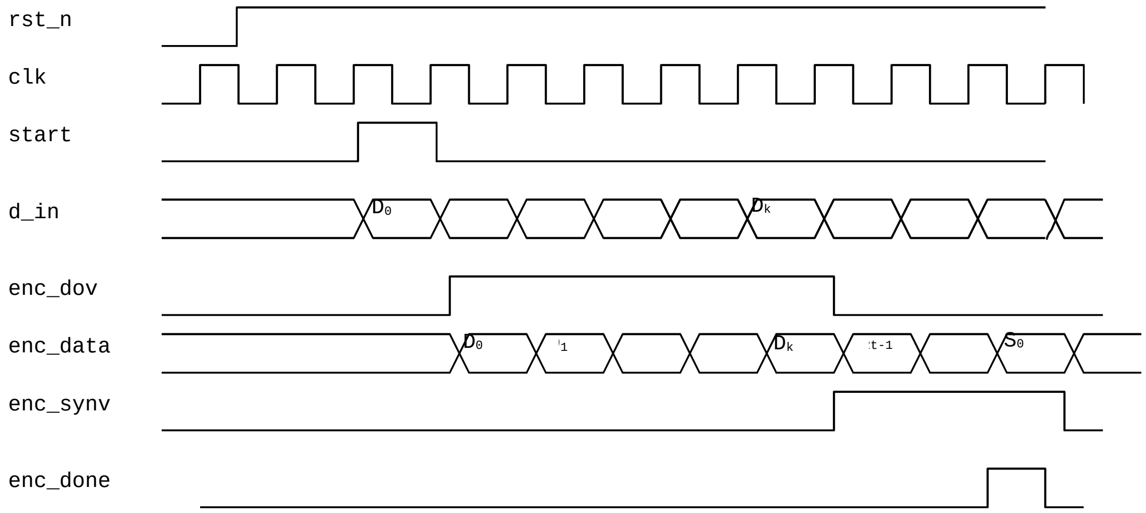
VHDL Entity Example

```
entity rs_encode is
  generic ( N  : integer := 255 ;
            K  : integer := 249
           );
    port( rst_n     : IN std_logic ;
          clk       : IN std_logic ;
          start     : IN std_logic ;
          d_in      : IN std_logic_vector(7 downto 0) ;
          enc_dout  : OUT std_logic_vector(7 downto 0);
          enc_dov   : OUT std_logic ;
          enc_synv  : OUT std_logic ;
          enc_done  : OUT std_logic
          ) ;
end rs_encode ;
```

Verilog Module Example

```
module rs_encode (
                  rst_n,
                  clk,
                  start,
                  d_in,
                  enc_dout,
                  enc_dov,
                  enc_synv,
                  enc_done
                 ) ;
```

Encoder Timing



6.2 RS Rx Syndrome Calculation

Port Signal Name	Input/Output	Function
rst_n	In	Resets core to initial state
clk	In	Clock
init	In	Reset Syndrome calculation to 0
d_in	In[x..0]	Data Input Bus
enb_dec	Out[x..0]	Enable Decoder
s0,s1,s2,…sN	Out	Syndrome outputs
syn_zero	Out	(s1,s1,s2,…sn) = 0

‘C’ Example

```
extern unsigned char msg_in[] ;
extern unsigned char enc_msg_out[] ;

void dec_syn_calc( n,
                   s1,
                   s2,
                   s3,
                   s4,
                   s5,
                   s6,
                   syn_zero )

dec_syn_calc(15,&s1,&s2,&syn_sero) ;
```

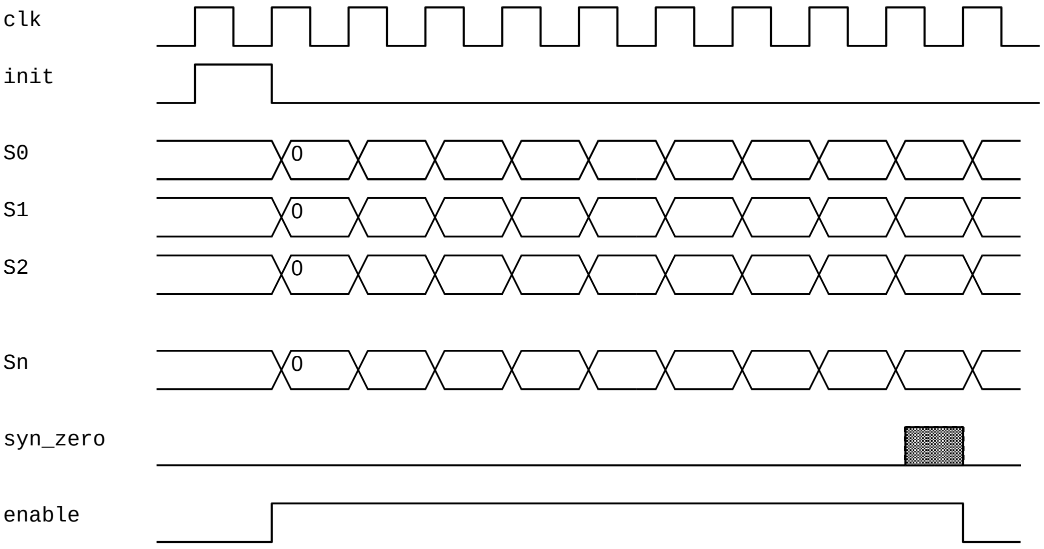
VHDL Example

```
entity dec_syn_calc is
   port( rst_n   : IN std_logic ;
         init    : IN std_logic ;
         clk     : IN std_logic ;
         enb_dec : IN std_logic ;
         d_in : IN std_logic_vector(7 downto 0) ;
         s1   : OUT std_logic_vector(7 downto 0) ;
         s2   : OUT std_logic_vector(7 downto 0) ;
         s3   : OUT std_logic_vector(7 downto 0) ;
         s4   : OUT std_logic_vector(7 downto 0) ;
         s5   : OUT std_logic_vector(7 downto 0) ;
         s6   : OUT std_logic_vector(7 downto 0) ;
         syn_zero : OUT std_logic
     ) ;
end dec_syn_calc ;
```

Verilog Example

```
module syn_calc( rst_n,
                  init,
                  clk,
                  enb_dec,
                  d_in,
                  s1,
                  s2,
                  s3,
                  s4,
                  s5,
                  s6,
                ) ;
```

Syndrome Calculation Timing



6.3 Berlkamp-Massey Algorithm

Port Signal Name	Input/Output	Function
rst_n	In	Resets core to initial state
clk	In	Clock
start_berl	In	Reset Syndrome calculation to 0
s0,s1,s2,…sN	In[x..0]	Syndrome Inputs
l0,l1,l2,…lN	In[x..0]	Error Locator Polynomial Outputs
Start_loc_search	Out	Start Error Location Search (Chien Search)

‘C’ Example

```
void berl_mas( int chk_sym, char *s, char *Lam, int dbg )

chk_sym : Number of symbols to correct in code word block.
*s : Pointer to syndromes this array should be 2 times the number of symbols to correct.
*Lam : Pointer to lamda polynomial array
dbg : Debug, 1= Print Diagnostic messages. 0 = Do not print diagnostic messages

berl_mas(1, rx_syndromes, error_locator_Lam, 0 )
```

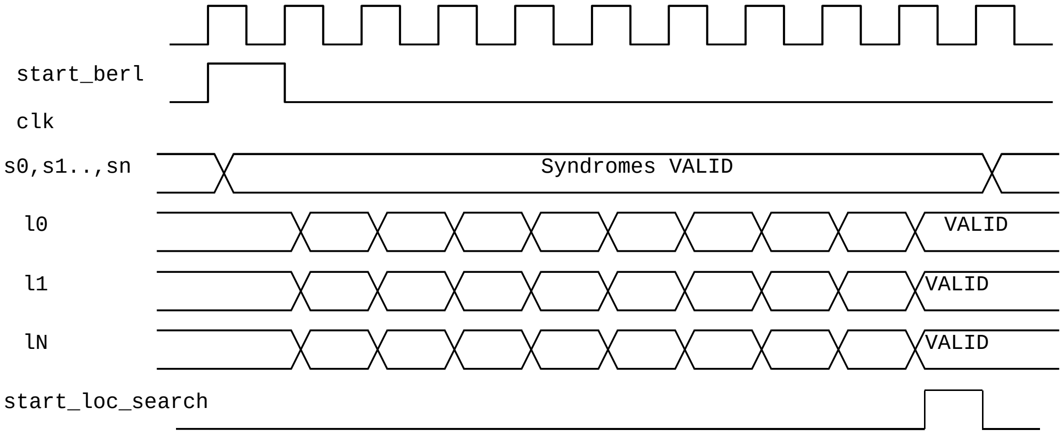
VHDL Example

```
entity berl_mas is
    port( rst_n : IN std_logic;
          clk : IN std_logic;
          start_berl : IN std_logic;
          s0: IN std_logic_vector(7 downto 0);
          s1: IN std_logic_vector(7 downto 0);
          s2: IN std_logic_vector(7 downto 0);
          s3: IN std_logic_vector(7 downto 0);
          s4: IN std_logic_vector(7 downto 0);
          s5: IN std_logic_vector(7 downto 0);
         l0: OUT std_logic_vector(7 downto 0);
         l1: OUT std_logic_vector(7 downto 0);
         l2: OUT std_logic_vector(7 downto 0);
         l3: OUT std_logic_vector(7 downto 0);
         l4: OUT std_logic_vector(7 downto 0);
         l5: OUT std_logic_vector(7 downto 0);
         start_loc_search : OUT std_logic
        ) ;
end berl_mas ;
```

Verilog Example

```
module berl_mas( rst_n,
                 clk,
                 start_berl,
                 s0,
                 s1,
                 s2,
                 s3,
                 s4,
                 s5,
                 l0,
                 l1,
                 l2,
                 l3,
                 l4,
                 l5,
                 start_loc_search
                ) ;
```

Berlkamp- Massey Error Polynomial Timing



6.4 RS Chien Search (Error Polynomial Root Location)

Port Signal Name	Input/Output	Function
rst_n	In	Resets core to initial state
clk	In	Clock
init_n	In	Reset
start_search	In	Start Error Location Root Search (Chien Search)
l0,l1,l2,…lN	In[x..0]	Error Locator Polynomial Inputs
lroot0,lroot1,lrootN	Out[x..0]	Error locations, 0,1,..,N
n_roots	Out[x..0]	Number of roots
search_done	Out	Search Done

‘C’ Example

```
void chien_search_hw( v, lam, r, lam_roots )
```

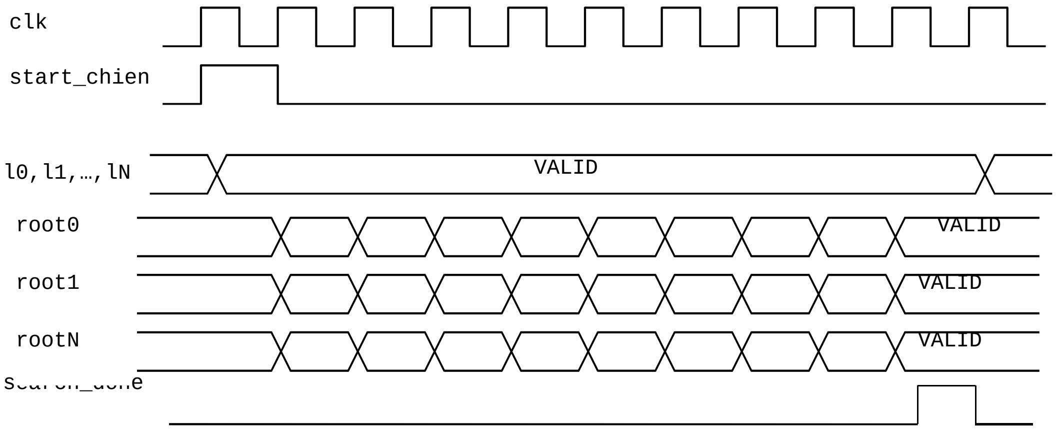
VHDL Example

```
entity chien_srch is
    port( rst_n : IN std_logic ;
          clk   : IN std_logic ;
          start_search : IN std_logic ;
          l0 : IN std_logic_vector(7 downto 0);
          l1 : IN std_logic_vector(7 downto 0);
          l2 : IN std_logic_vector(7 downto 0);
          l3 : IN std_logic_vector(7 downto 0);
          l4 : IN std_logic_vector(7 downto 0);
          l5 : IN std_logic_vector(7 downto 0);
         lroot0 : OUT std_logic_vector(7 downto 0);
         lroot1 : OUT std_logic_vector(7 downto 0);
         lroot2 : OUT std_logic_vector(7 downto 0);
         n_roots : OUT std_logic_vector(2 downto 0);
         search_done  : OUT std_logic
       ) ;
end chien_srch ;
```

Verilog Example

```
module chien_srch( rst_n,
                  clk,
                  start_search,
                  l0,
                  l1,
                  l2,
                  l3,
                  l4,
                  l5,
                  lroot0,
                  lroot1,
                  lroot2,
                  n_roots,
                  search_done
                ) ;
```

Chien Search Root Calculation Timing



6.5 RS Forney – Error Correction Values

Port Signal Name	Input/Output	Function
rst_n	In	Resets core to initial state
clk	In	Clock
init_n	In	Reset
start_forney	In	Start Error Location Root Search (Chien Search)
s0,s1,s2,…sN	In[x..0]	Syndrome Inputs
l0,l1,l2,…lN	In[x..0]	Error Locator Polynomial Inputs
lroot0,lroot1,lrootN	Out[x..0]	Error locations, 0,1,..,N
n_roots	Out[x..0]	Number of roots
corr_ewrr_val0, corr_err_val1,…, corr_err_valN	Out[x..0]	Correction Values at root0, root1,…,rootN
omega_dlam_rdy	Out	
search_done	Out	Search Done

'C' Example

```
void forney(v, syn, lam, n_roots, lam_root, corr_val, dbg )
```

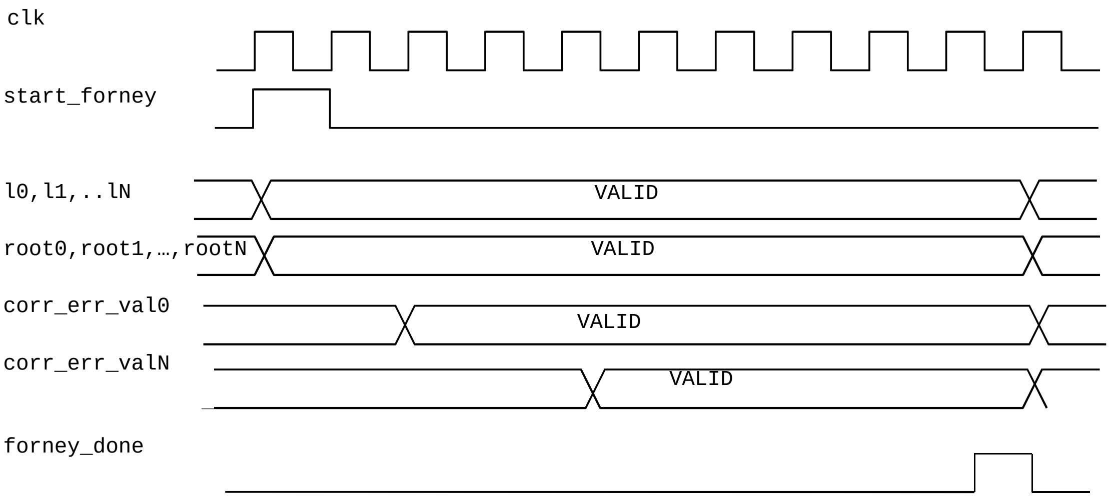
VHDL Example

```
entity forney is
   port ( rst_n : IN std_logic;
          clk : IN std_logic;
          start_forney : IN std_logic;
           root_search_done : IN std_logic;
          s0 : IN std_logic_vector(7 downto 0);
          s1 : IN std_logic_vector(7 downto 0);
          s2 : IN std_logic_vector(7 downto 0);
          s3 : IN std_logic_vector(7 downto 0);
          s4 : IN std_logic_vector(7 downto 0);
          s5 : IN std_logic_vector(7 downto 0);
          l0 : IN std_logic_vector(7 downto 0);
          l1 : IN std_logic_vector(7 downto 0);
          l2 : IN std_logic_vector(7 downto 0);
          l3 : IN std_logic_vector(7 downto 0);
          l4 : IN std_logic_vector(7 downto 0);
          l5 : IN std_logic_vector(7 downto 0);
         lroot0 : IN std_logic_vector(7 downto 0);
         lroot1 : IN std_logic_vector(7 downto 0);
         lroot2 : IN std_logic_vector(7 downto 0);
         n_roots : IN std_logic_vector(2 downto 0);
         corr_err_val0 : OUT std_logic_vector(7 downto 0);
         corr_err_val1 : OUT std_logic_vector(7 downto 0);
         corr_err_val2 : OUT std_logic_vector(7 downto 0);
         omega_dlam_rdy : OUT std_logic;
         forney_done : OUT std_logic
        ) ;
```

Verilog Example

```
module forney( rst_n,
              clk,
              start_forney,
               root_search_done,
              s0,
              s1,
              s2,
              s3,
              s4,
              s5,
              l0,
              l1,
              l2,
              l3,
              l4,
              l5,
             lroot0,
             lroot1,
             lroot2,
             n_roots,
             corr_err_val0,
             corr_err_val1,
             corr_err_val2,
             omega_dlam_rdy,
             forney_done
            ) ;
```

Forney Error Values Calculation Timing



6.6 ROM Lookup Tables

ROM look-up tables are used to multiply two field elements, divide two field elements, or exponential multiply of a field element. Division is performed by using the inverse of multiplication…1/a^n = a ^-n and adding the two field elements. The ROM tables are all the field elements of gf(2^n) a^n, 1/a^n and (a^n)^n where n is the index of the field valued. RSTK produces simulation support files base 16 (*.hex) for the simulation of the RS code produced. Altera or Xilinx FPGAs memory initialization files are also produced (*.mif, *.coe, *.ucf).

7 RS(15,9) Example

For a RS code GF(2^4) with the number of symbols to correct t=3, the number of check symbols v=2*3=6, and a generator base of 1.

The field polynomial is:

$$P(x) = x^4 + x^1 + 1 = 10011$$

The generator polynomial is:

$$G(x) = (x+\alpha^1)(x+\alpha^2)(x+\alpha^3)(x+\alpha^4)(x+\alpha^5)(x+\alpha^6)$$

$$= x^6 + \alpha^{10}x^5 + \alpha^{14}x^4 + \alpha^4x^3 + \alpha^6x^2 + \alpha^9x^1 + \alpha^6$$

The parameters on the command line are:

VHDL

```
rstk –px4 –n4 –c3 –b1 -gvdhl
```

Verilog

```
rstk –px4 –n4 –c3 –b1 –gvlog
```

C

```
rstk –px4 –n4 –c3 –b1 –gC
```

The output to the console screen is:

```
Reed Solomon Polynomial: x4
GF Field Size: 4
Number of check symbols: 3
Generator Polynomial Base: 1
Code Generator: C
FPGA Manufacturer: xlx
ROM Type: sim
g(0) = a^6  pwsp[0] = c
 g(1) = a^9  pwsp[1] = a
 g(2) = a^6  pwsp[2] = c
 g(3) = a^4  pwsp[3] = 3
 g(4) = a^14  pwsp[4] = 9
 g(5) = a^10  pwsp[5] = 7
 g(6) = a^15  pwsp[6] = 1

NOTE: a^15 = a^1 in GF(16)
```

7.1 RS Codec Output Files

The files produced for the selected RS code are

‘C’
gf_arith.c
rs_enc.c
rs_syndec.c
rs_berl.c
rs_chien.c
rs_forney.c
poly_math.c

VHDL
gf_arith.vhd
rs_enc.vhd
rs_syndec.vhd
rs_berl.vhd
rs_chien.vhd
rs_forney.vhd
poly_reg.vhd

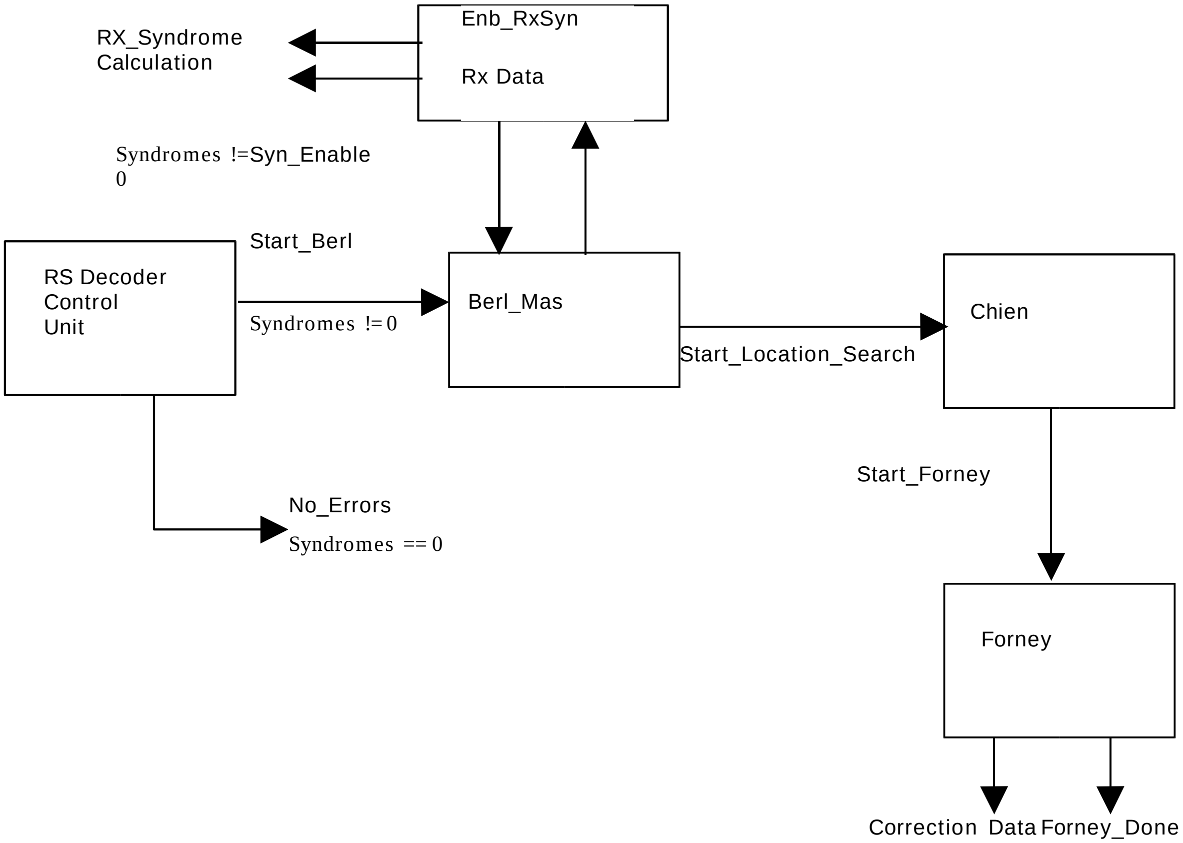
VLOG
gf_arith.v
rs_enc.v
rs_syndec.v
rs_berl.v
rs_chien.v
rs_forney.v
poly_reg.v

The ROM tables/files produced are

```
rom_c_tables.c
gf_alpha_rom.hex
gf_invalpha_rom.hex
gf_powalpha_rom.hex
```

7.2 Building a RS Decoder

The architectural design of the decoder core modules allows for simple interconnetion between them to assemble a RS decoder core.



* The RS decoder control module shown above must be supplied to complete the core and is very dependent on the user implementation.

7.3 References

Theory and Practice of Error Control Codes
Richard E. Blahut, Copyright 1983 Addison&Wesley

Practical Error Correction Design for Engineers
Neal Glover and Trent Dudley, Cirrus Logic