Reed Solomon Encoder C Model (Comment # 234)
Martin Langhammer
September 2012
Need

- Generator Polynomial and example codeword previously presented
  - Can publish encoder source code in Annex as well
    - Concise - approximately 1pp
- Reed Solomon encoder model requested by non-FEC expert for system simulation
  - Polynomial and codeword not sufficient
- Interoperability verification
  - Everybody can generate codewords
Model Background

- **Altera release to public domain**
  - Free use in any technology for any purpose
- **Pedantic algorithmic**
  - Based on well known methods
  - No hardware implementation details
  - Not optimized for software
- **PAM-2 and PAM-4 examples**
  - Included generator polynomials
- **Available now on request**

```c
//*** Global variables for PAM-2 (528,514,7,10) code - clause 91 802.3bj ***
long n_symbols = 528;
long k_symbols = 514;
long polynomial = 1033;
unsigned long generator_polynomial[1024] = {904,6,701,32,656,925,900,614,391,592,265,945,290,432};

//*** Global variables for PAM-4 (544,514,15,10) code - clause 91 802.3bj ***
long n_symbols = 544;
long k_symbols = 514;
long polynomial = 1033;
unsigned long generator_polynomial[1024] = {575,552,167,235,552,1,106,585,232,193,539,132,84,720,495,385,
562,505,650,596,516,561,532,127,193,158,128,634,325};

long check_symbols;
unsigned long codeword[1024];
unsigned long parity[1024];
unsigned long multiply(long aa, long bb)
{
  unsigned long expand = 0;
  long k;
  for (k=0;k<10;k++)
  {
    if (bb & (1 << k))
      expand = expand ^ (aa << k);
  }
  for (k=0;k<9;k++)
  {
    if ((expand >> (18-k)) & 1)
      expand = expand ^ (polynomial << (8-k));
  }
  return expand;
}

void encode()
{
  long k,j;
  unsigned long multiplier;
  unsigned long generator_vector[1024];
  unsigned long encoder_divide[1024];
  for (k=0;k<check_symbols;k++)
    encoder_divide[k] = 0;
  for (k=0;k<k_symbols;k++)
  {
    multiplier = codeword[k] ^ encoder_divide[0];
    for (j=0;j<check_symbols;j++)
      generator_vector[j] = multiply(multiplier,generator_polynomial[j]);
    for (j=0;j<check_symbols-1;j++)
      encoder_divide[j] = generator_vector[j] ^ encoder_divide[j+1];
    encoder_divide[check_symbols-1] = generator_vector[check_symbols-1];
  }
  for (j=0;j<check_symbols;j++)
    codeword[check_symbols+k] = encoder_divide[j];
}

void main()
{
  long k;
  check_symbols = n_symbols - k_symbols;
  **** Generate simple codeword data symbols ****
  for (k=0;k<k_symbols;k++)
    codeword[k] = 1023-k;
  encode();
  for (k=0;k<check_symbols;k++)
    printf(“%ld ”,codeword[k]);
}
```
Details – Generator Polynomials

- PAM-2 and PAM-4 polynomials
  - Any other polynomial can be manually entered

```c
/**
 * *** Global variables for PAM-2 (528,514,7,10) code - clause 91 802.3bj ***
 */
long n_symbols = 528;
long k_symbols = 514;
long polynomial = 1033;
unsigned long generator_polynomial[1024] =
{904,6,701,32,656,925,900,614,391,592,265,945,290,432};

/**
 * *** Global variables for PAM-4 (544,514,15,10) code - clause 91 802.3bj ***
 */
long n_symbols = 544;
long k_symbols = 514;
long polynomial = 1033;
unsigned long generator_polynomial[1024] =
{575,552,187,230,552,1,108,565,282,249,593,132,94,720,495,385,
  942,503,883,361,788,610,193,392,127,185,158,128,834,523};

long check_symbols;
unsigned long codeword[1024];
unsigned long parity[1024];
```
Details – GF() Multiplier

- Expansion and Reduction algorithm
  - Fixed at 10 bits in model, but could be changed for other codes

```c
unsigned long multiply (long aa, long bb)
{
    unsigned long expand = 0;
    long k;

    for (k=0;k<10;k++)
    {
        if (bb & (1 << k))
            expand = expand ^ (aa << k);
    }

    for (k=0;k<9;k++)
    {
        if ((expand >> (18-k)) & 1)
            expand = expand ^ (polynomial << (8-k));
    }

    return expand;
}
```
Details – Polynomial Division

- Encoding loop

```c
void encode()
{
    long k, j;
    unsigned long multiplier;
    unsigned long generator_vector[1024];
    unsigned long encoder_divide[1024];

    for (k = 0; k < check_symbols; k++)
        encoder_divide[k] = 0;

    for (k = 0; k < k_symbols; k++)
    {
        multiplier = codeword[k] ^ encoder_divide[0];

        for (j = 0; j < check_symbols; j++)
            generator_vector[j] = multiply(multiplier, generator_polynomial[j]);

        for (j = 0; j < check_symbols - 1; j++)
            encoder_divide[j] = generator_vector[j] ^ encoder_divide[j + 1];
        encoder_divide[check_symbols - 1] = generator_vector[check_symbols - 1];

        for (j = 0; j < check_symbols; j++)
            codeword[j + k_symbols] = encoder_divide[j];
    }
}
```
Details – Data Symbols

- for{} loop generates previously presented example codeword
  - Put desired data symbols in first k symbols of unsigned long codeword[]

```c
void main()
{
    long k;

    check_symbols = n_symbols - k_symbols;

    //*** Generate simple codeword data symbols ***
    for (k=0;k<k_symbols;k++)
        codeword[k] = 1023-k;

    encode();

    for (k=0;k<n_symbols;k++)
        printf("%ld ",codeword[k]);
}
```
Decoder

- Public domain decoder also completed
- Tested with PAM-2 and PAM-4 decoders
- Like encoder, based on well known algorithms
  - No hardware implementation details
  - Not optimized for software
Thank You