Small World Communications

# SCE02C CCSDS SCCC Encoder

28 December 2023 (Version 1.00) Product Specification

## SCE02C Features

- 4 state CCSDS compatible serially concatenated convolutional code (SCCC) encoder
- Code rates from 0.355 to 0.899
- Data lengths from 5758 to 43678 bits
- Interleaver sizes from 8640 to 65520 bits
- Symbol interleaver sizes from 16200 to 48600 bits with QPSK, 8PSK, 16APSK, 32APSK or 64APSK modulation and 8100 coded symbols
- Optional 6–bit coded symbol or 12–bit inphase (I) and quadrature (Q) output
- Includes 256 symbol frame marker, 64 symbol frame descriptor and optional pilot symbols with programmable I and Q pseudo randomiser
- Continuous coded symbol data out
- Up to 575 MHz internal clock
- Up to 1.14 Gbit/s encoding speed with 213 MHz symbol clock
- 1191 6–input LUTs and 30 18kB RAMB18s
- Asynchronous logic free design
- Available as EDIF and VHDL core for Xilinx FPGAs under SignOnce IP License. ASIC, Intel/Altera, Lattice and Microsemi/Actel cores available on request.

## Introduction

The SCE02C is a 4 state systematic recursive CCSDS [1] compatible SCCC encoder. Twenty different interleaver sizes from 8640 to 65520 bits are implemented. Twenty seven different code rates $R$ from 0.355 to 0.899 can be selected, giving bandwidth efficiencies from 0.711 to 5.392 bit/sym, excluding header and pilot symbols.

The outer rate 1/2 code is punctured to a rate of 2/3. The outer four bit tail punctured to three bits. The inner rate 1/2 code uses different forms of puncturing for the systematic and parity bits, with the four bit tail not punctured. The interleaver size is of size $I = 1.5K+3$, where $K$ is the number of information bits, which ranges from 5758 to 43678.

The number of coded bits is $8100m$, where $m$ is the number of bits per symbol, ranging from $m = 2$ for QPSK to 6 for 64APSK. To allow a continuous output stream, ping–pong interleaver and symbol memories are used to buffer the input data to be encoded.

The encoded stream consists of a 256 bit $\pi/2$–BPSK modulated frame marker, a 64 bit $\pi/2$–BPSK modulated frame descriptor and 16 codeword segments (CWS). Each CWS consists of 8100 coded symbols using either QPSK (quadrature phase shift keying), 8PSK, 16APSK (amplitude phase shift keying), 32APSK or 64APSK modulation. If pilot symbols are selected, each CWS has a total of 240 pilot symbols, distributed through the CWS in 15 groups of 16 pilot symbols.

The CWS symbols (including the pilots) are I and Q scrambled using a programmable pseudo randomiser, initialised to one of $2^{18}–1 = 262143$ values.

Figure 1 shows the schematic symbol for the SCE02C encoder. The EDIF core can be used with Xilinx Foundation or Integrated Software Environment (ISE) software. The VHDL core can be used with Xilinx ISE or Vivado software. Custom VHDL cores can be used in ASIC designs.

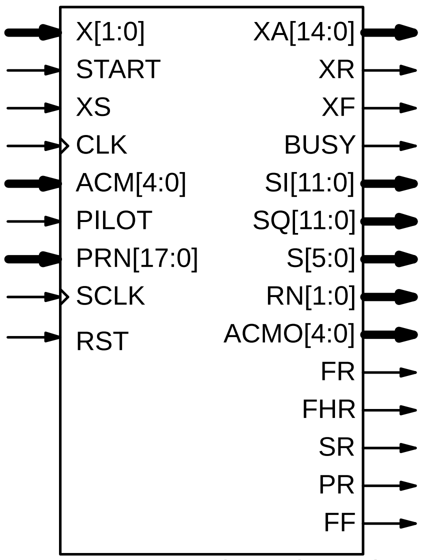

Figure 1: SCE02C schematic symbol.

Table 1 shows the performance achieved for various Xilinx parts with $K = 43678$ (ACM = 27). $T_{cp}$ is the minimum clock period over recommended operating conditions. $f_S$ is the maximum coded symbol rate. These performance figures may change due to device utilisation and configuration. Note that Zynq devices up to XC7Z020 and from XC7Z030 use programmable logic equivalent to Artix–7 and Kintex–7 devices, respectively.

**Table 1: Example performance**

| Part | $T_{cp}$ (ns) | Speed (Mbit/s) | $f_s$ (Msym/s) |
|---|---|---|---|
| XC7S25–1 | 5.233 | 381 | 70.8 |
| XC7S25–2 | 4.283 | 466 | 86.5 |
| XC7A15T–1 | 5.362 | 372 | 69.1 |
| XC7A15T–2 | 4.445 | 449 | 83.3 |
| XC7A15T–3 | 3.935 | 507 | 94.1 |
| XC7K70T–1 | 3.945 | 506 | 93.9 |
| XC7K70T–2 | 3.340 | 598 | 110.9 |
| XC7K70T–3 | 3.003 | 665 | 123.4 |
| XCKU035–1 | 3.494 | 572 | 106.0 |
| XCKU035–2 | 3.027 | 660 | 122.4 |
| XCKU035–3 | 2.467 | 810 | 150.2 |
| XCKU3P–1 | 2.186 | 914 | 169.5 |
| XCKU3P–2 | 1.928 | 1036 | 192.2 |
| XCKU3P–3 | 1.739 | 1149 | 213.1 |

## Signal Descriptions

ACM Advanced Coding and Modulation Format (1 to 27)
ACMO ACM Output (1 to 27)
BUSY Encoder Busy (new data not accepted)
CLK Encoder Clock
FHR Frame Header Ready
FR Frame Ready
FF Frame Finish
PILOT Pilot Select
0 = No pilot symbols
1 = Pilot symbols inserted
PR Pilot Ready
PRN Pseudo Randomiser Number (1 to 262143)
RN Randomiser for I and Q (0 to 3)
RST Synchronous Reset
S Symbol (0 to 63)
SCLK Symbol Clock
SI Symbol In–phase (12–bit 2's comp.)
SR Symbol Ready
START Encoder Start
SQ Symbol Quadrature (12–bit 2's comp.)
X Data In
XA Data In Address (0 to 43677)
XF Data in Finish
XR Data In Ready

## Encoder Operation

Figure 2 gives a simplified block diagram of the SCE02C CCSDS SCCC encoder.

### Outer Encoder

To increase encoding speed, the outer rate 1/2 four state systematic recursive convolutional encoder that is punctured to a rate of 2/3, is converted to a non–punctured rate 2/3 encoder. For the standard rate 1/2 code, we have

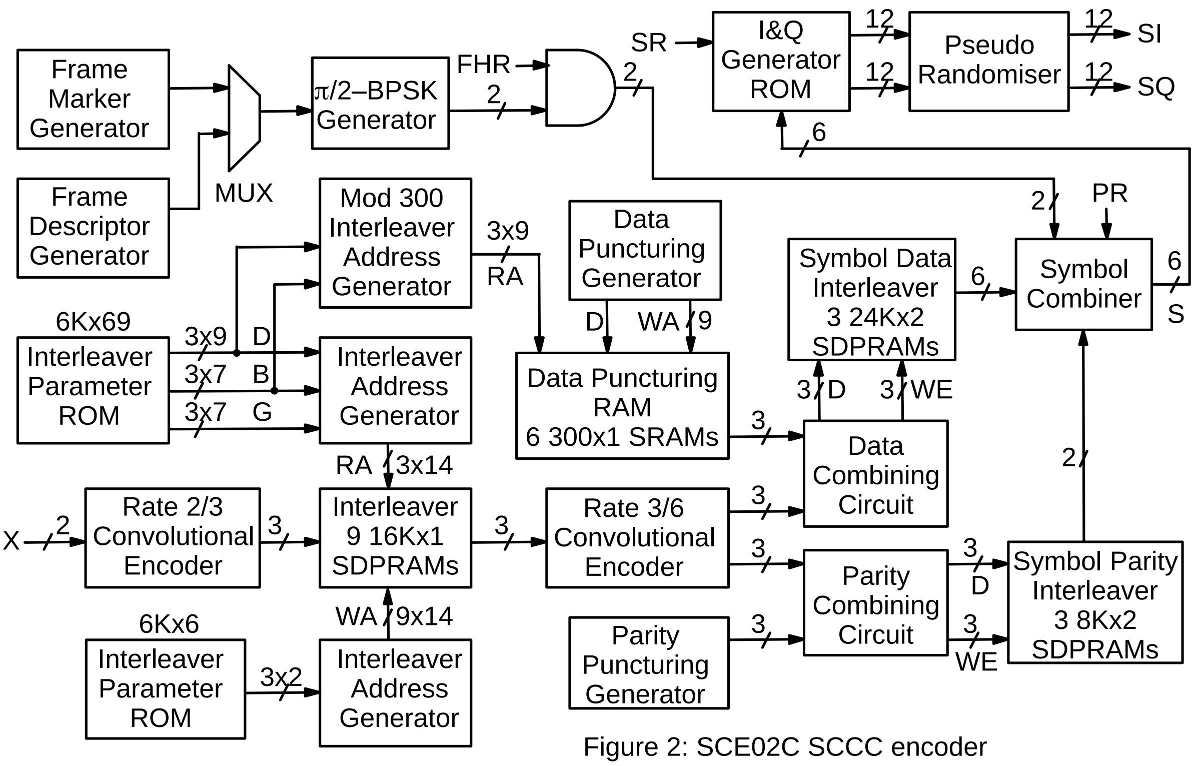

Figure 2: SCE02C SCCC encoder

$$\begin{aligned} y_i^0 &= x_i, \\ y_i^1 &= x_i + s_i^0, \\ s_{i+1}^0 &= x_i + s_i^0 + s_i^1, \\ s_{i+1}^1 &= s_i^0, \end{aligned} \tag{1}$$

where $x_i$ is the input data bit for $0 \le i \le K-1$, $y_i^0$ and $y_i^1$ are the two coded bits, $s_i^0$ and $s_i^1$ are the two bits corresponding to the state of the encoder and + represents modulo–2 (XOR) addition. For $K \le i \le K+1$, the tail is generated with

$$\begin{aligned} y_i^0 &= s_i^0 + s_i^1, \\ y_i^1 &= s_i^1, \\ s_{i+1}^0 &= 0, \\ s_{i+1}^1 &= s_i^0. \end{aligned} \tag{2}$$

Incrementing $i$ by one, we have for the main data

$$\begin{aligned} y_{i+1}^0 &= x_{i+1}, \\ y_{i+1}^1 &= x_{i+1} + s_{i+1}^0 = x_{i+1} + x_i + s_i^0 + s_i^1, \\ s_{i+2}^0 &= x_{i+1} + s_{i+1}^0 + s_{i+1}^1 = x_{i+1} + x_i + s_i^1, \\ s_{i+2}^1 &= s_{i+1}^0 = x_i + s_i^0 + s_i^1, \end{aligned} \tag{3}$$

and for the tail we have

$$\begin{aligned} y_{i+1}^0 &= s_{i+1}^0 + s_{i+1}^1 = s_i^0, \\ y_{i+1}^1 &= s_{i+1}^1 = s_i^0, \\ s_{i+2}^0 &= 0, \\ s_{i+2}^1 &= s_{i+1}^0 = 0. \end{aligned} \tag{4}$$

Letting $i = 2j$, we have $x_j^0 = x_i$, $x_j^1 = x_{i+1}$, $y_j^0 = y_i^0$, $y_j^1 = y_i^1$, $y_j^2 = y_{i+1}^0$, $s_j^0 = s_i^0$, $s_j^1 = s_i^1$, $s_{j+1}^0 = s_{i+2}^0$ and $s_{j+1}^1 = s_{i+2}^1$, for $0 \le j \le K/2$. There is no $y_j^3$ since $y_{i+1}^1$ is punctured. This gives the equations for the rate 2/3 encoder for $0 \le j \le K/2-1$ as

$$\begin{aligned} y_j^0 &= x_j^0, \\ y_j^1 &= x_j^0 + s_j^0, \\ y_j^2 &= x_j^1, \\ s_{j+1}^0 &= x_j^1 + x_j^0 + s_j^1, \\ s_{j+1}^1 &= x_j^0 + s_j^0 + s_j^1, \end{aligned} \tag{5}$$

and for the tail with $j = K/2$ as

$$\begin{aligned} y_j^0 &= s_j^0 + s_j^1, \\ y_j^1 &= s_j^1, \\ y_j^2 &= s_j^0. \end{aligned} \tag{6}$$

## Interleaver

The interleaver needs to write three consecutive bits into the RAM and then read three bits in interleaved order. This is achieved by the splitting the RAM into nine separate simple dual port random access (SDPRAM) memories, each of size 16Kx1. We let coded bit $y_j^h$, $0 \le h \le 2$ be written to one of the nine memories indicated by the pair $h,f(3j+h)$ where

$$f(i) = \pi^{-1}(i) \bmod 3 \tag{7}$$

where $\pi^{-1}(i)$, $0 \le i \le I-1$ is the inverse function of the interleaver function

$$\begin{aligned} \pi(i) &= W[(i^W + \beta(i_W)) \bmod 120] + \alpha(i_W), \\ i_W &= i \bmod W, \\ i^W &= i \operatorname{div} W = (i - i_W)/W, \\ W &= I/120 = (K+2)/80. \end{aligned} \tag{8}$$

We have $0 \le \alpha(i_W) \le W-1$ and $0 \le \beta(i_W) \le 119$ are lookup table constants dependent the interleaver size $I$. We can express the inverse as

$$\pi^{-1}(i) = W[(i^W - \beta(\alpha^{-1}(i_W))) \bmod 120] + \alpha^{-1}(i_W), \tag{9}$$

where $\alpha^{-1}(i)$, $0 \le i \le W-1$ is the inverse function of $\alpha(i)$. Since $W$ is a multiple of 3 we thus have

$$f(i) = \alpha^{-1}(i \bmod W) \bmod 3. \tag{10}$$

Since $i = Wi^W + i_W$, we have $f(i) = f(i_W)$ and thus only need a lookup table for $f(i_W)$, $0 \le i_W \le W-1$. The lookup table will output three 2–bit values for $i_W = 3j+h$, $0 \le j \le W/3-1$. Each bit $y_j^h$, will be written to RAM $h,f(3j+h)$. Each RAM also has an individual write address counter, which is incremented after each write. Due to the random nature of the $h,f(3j+h)$ pair, the number of writes to each RAM will be uneven.

Table 2 shows the number of writes for each RAM for each of the 19 interleavers. The maximum value is 67 for ACM = 26 and RAM 1,2. To simplify the calculation of the RAM write address, which must be incremented by the maximum address value every $W/3$ clock cycles for 120 rows, we let the increment value be equal to 68. The total address space is then 120x68 = 8160. As a ping–pong interleaver is used (writing to one half while reading from the other half), each of the nine RAMs is of size 16320x1, which is implemented using a 16Kx1 BlockRAM.

The summation of all the different $W$ values is $W_{\text{sum}} = 5367$. The total address is space for the ROM is thus $W_{\text{sum}}/3 \times 3\lceil \log_2 3 \rceil$ or 1789x6 which is implemented using a 2Kx9 BlockRAM.

For the inner encoder, three bits are read at a time in interleaved order. Bit $z_i^h$, $0 \le i \le I/3-1$, $0 \le h \le 2$ is read from RAM $g(3i+h),h$ where

$$g(i) = \pi(i) \bmod 3. \tag{11}$$

Similar to $f(i)$, we have that

$$g(i) = \alpha(i \bmod W) \bmod 3. \tag{12}$$

Thus, the three bits $z_j^h$, $0 \le h \le 2$, will be read from RAM $g(3j+h),h$, $0 \le j \le W/3-1$. As the column addresses are unevenly distributed across

**Table 2: Number of writes for RAM *h*,*f*(3*j*+*h*) and clock factor *F***

| ACM | $K$ | $I$ | $W$ | 0,0 | 0,1 | 0,2 | 1,0 | 1,1 | 1,2 | 2,0 | 2,1 | 2,2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 5758 | 8640 | 72 | 10 | 8 | 6 | 6 | 10 | 8 | 8 | 6 | 10 |
| 2 | 6958 | 10440 | 87 | 7 | 11 | 11 | 9 | 9 | 11 | 13 | 9 | 7 |
| 3 | 8398 | 12600 | 105 | 14 | 11 | 10 | 10 | 10 | 15 | 11 | 14 | 10 |
| 4 | 9838 | 14760 | 123 | 11 | 13 | 17 | 16 | 15 | 10 | 14 | 13 | 14 |
| 5,7 | 11278 | 16920 | 141 | 18 | 16 | 13 | 14 | 15 | 18 | 15 | 16 | 16 |
| 6,8 | 13198 | 19800 | 165 | 20 | 19 | 16 | 17 | 17 | 21 | 18 | 19 | 18 |
| 9 | 14878 | 22320 | 186 | 20 | 26 | 16 | 21 | 20 | 21 | 21 | 16 | 25 |
| 10 | 17038 | 25560 | 213 | 24 | 29 | 18 | 24 | 20 | 27 | 23 | 22 | 26 |
| 11,13 | 19198 | 28800 | 240 | 30 | 23 | 27 | 23 | 31 | 26 | 27 | 26 | 27 |
| 12,14 | 21358 | 32040 | 267 | 28 | 28 | 33 | 27 | 36 | 26 | 34 | 25 | 30 |
| 15 | 23518 | 35280 | 294 | 31 | 34 | 33 | 39 | 35 | 24 | 28 | 29 | 41 |
| 16,18 | 25918 | 38880 | 324 | 35 | 37 | 36 | 36 | 40 | 32 | 37 | 31 | 40 |
| 17,19 | 28318 | 42480 | 354 | 36 | 45 | 37 | 38 | 33 | 47 | 44 | 40 | 34 |
| 20 | 30958 | 46440 | 387 | 38 | 43 | 48 | 42 | 43 | 44 | 49 | 43 | 37 |
| 21,23 | 33358 | 50040 | 417 | 50 | 48 | 41 | 42 | 47 | 50 | 47 | 44 | 48 |
| 22,24 | 35998 | 54000 | 450 | 52 | 49 | 49 | 49 | 48 | 53 | 49 | 53 | 48 |
| 25 | 38638 | 57960 | 483 | 55 | 51 | 55 | 53 | 57 | 51 | 53 | 53 | 55 |
| 26 | 41038 | 61560 | 513 | 64 | 62 | 45 | 46 | 58 | 67 | 61 | 51 | 59 |
| 27 | 43678 | 65520 | 546 | 64 | 65 | 53 | 63 | 53 | 66 | 55 | 64 | 63 |

the nine RAMs, we can't directly use $\alpha(i)$. Instead, we use $\gamma(i)$ which is determined using the following algorithm, where the array alpha[*i*] corresponds to $\alpha(i)$, alphai[*i*] to $\alpha^{-1}(i)$, column[*h*,*f*] to the number of writes to RAM *h*,*f* and gamma[*i*] to $\gamma(i)$, which is the column for $\alpha(i)$.

```
for I := 0 to W–1 do
  alphai[alpha[I]] := I;
for H := 0 to 2 do
  for F := 0 to 2 do
    column[H,F] := 0;
for I := 0 to W–1 do
  begin{count}
    H := I mod 3;
    F := alphai[I] mod 3;
    gamma[alphai[I]] := column[H,F];
    column[H,F] := column[H,F]+1;
  end;{count}
```

For each $0 \le j \le W/3-1$, where $j = (i \bmod W)$ div 3 and $h = i \bmod 3$, the ROM outputs three values of $\gamma(3j+h)$, $0 \le h \le 2$. The maximum values of $\gamma(3j)$, $\gamma(3j+1)$ and $\gamma(3j+2)$ are 63, 64 and 66, respectively. Thus, we could use 6 bits for $h = 0$ and 7 bits for $h = 1$ and 2. However, as this does not lead to a reduction in the number of BlockRAMs, we simplify this to using 7 bits for all three values.

The three read address for $0 \le h \le 2$ are formed for $0 \le j \le W/3-1$ from

$$\mathrm{RA}(i) = 68[i^W + \beta(3j + h)) \bmod 120] + \gamma(3j + h) \tag{13}$$

where $0 \le i \le I-1$, $j = (I \bmod W)$ div 3 and $h = i \bmod 3$. As the maximum value of $\beta(i)$ is 119, this requires 7 bits for each of the three values that are output.

In order to select RAM $g(3j+h)$,$h$ we need to also output the three values for $g(3j+h)$, which is similar to $f(3j+h)$ requires a total of 3x2 = 6 bits. As shown in the next section, we calculate this from the ROM values used in puncturing the systematic data.

## Data Puncturing

The standard uses a fairly complex scheme for puncturing the systematic data of the inner convolutional encoder. We first need to generate a length 300 lookup table containing a puncturing pattern of 0's and 1's. As we encode three bits at a time, we need three of these lookup tables. Also, to allow the code rate to change from one frame to the next, we write the new pattern into one half of a 600x1 RAM while outer encoding, and read from the other half of the RAM while inner encoding. Thus, we need a total of six 300x1 RAMs.

For each ACM value, the standard specifies that $S_{sur}$ bits of the 300 will not be punctured. Values from $S_{sur}$ = 300 for ACM = 1 and 2 to 208 for ACM = 27 are provided. Another table is provided where for each $S_{sur}$ value from 299 down to 200, the puncturing position is given, e.g., the positions for $S_{sur}$ = 299 and 298 are 76 and 1, respectively. The puncturing positions are formed from address 299 down to $S_{sur}$. If $S_{sur}$ = 300, no bits are punctured.

To create the puncturing pattern, we first write 300 one's into the puncturing RAM from address 0 to 299. For the ACM value, we use a small LUT to output $S_{pun}$ = 299–$S_{sur}$, which ranges from –1 to 91. If $S_{pun}$ = –1 (represented by 127 from the LUT) we do not write any zeros. For $S_{pun} \geq 0$, we use a counter that increments from 0 to $S_{pun}$, writing $S_{pun}$+1 zeros to the addresses given from another LUT, e.g., we write zeros to addresses 76 and 1 for counter addresses 0 and 1.

We have the address $a(k,l)$, $k = i$ div $W$, $l = i$ mod W, $0 \leq i \leq I–1$, to the puncturing RAM is

$$a(k, l) = \pi(k, l) \bmod 300 \tag{14}$$

where

$$\pi(k, l) = W((k + \beta(l)) \bmod 120) + \alpha(l). \tag{15}$$

To simplify the calculation of $a(k,l)$ we use differences. Due to the use of lookup tables for $\alpha(l)$ and $\beta(l)$ we let the first $W$ initial values for $0 \leq l \leq W–1$ be

$$a(0, l) = [(W\beta(l) \bmod 120) + \alpha(l)] \bmod 300. \tag{16}$$

We have the difference as

$$\pi(k+1, l) - \pi(k, l) = \begin{cases} W \; ; \; k+1+\beta(l) < 120, \\ -119W \; ; \; k+1+\beta(l) = 120, \\ W \; ; \; k+1+\beta(l) > 120. \end{cases} \tag{17}$$

Thus, we can calculate the next values as

$$a(k+1, l) = \begin{cases} (a(k, l) + W) \bmod 300 \; ; \; k+\beta(l) \neq 119, \\ (a(k, l) - 119W) \bmod 300 \; ; \; k+\beta(l) = 119. \end{cases} \tag{18}$$

As we already have $\beta(l)$, we only need $\delta(l) = a(0,l)$ from the ROM. The values of $W$ mod 300 and –119$W$ mod 300 are stored in LUTs and are used for the additions.

As we need to generate three bits in parallel, we let $l = 3j+h$, $j = l$ div 3 and $h = l$ mod 3, where we output $\delta(3j+h)$ for $0 \leq h \leq 2$ in parallel for $0 \leq j \leq W/3–1$. As the maximum value of $\delta(l)$ is 299, this requires three 9 bit values to be stored.

As we have

$$\begin{aligned} \delta(l) \bmod 3 &= (\pi(0, l) \bmod 300) \bmod 3 \\ &= [\pi(0, l) - 300(\pi(0, l) \text{ div } 300)] \bmod 3 \\ &= \pi(0, l) \bmod 3 = \alpha(l) \bmod 3 = g(l) \end{aligned} \tag{19}$$

we can use $\delta(l)$ to calculate which RAM to read from, thus reducing the size of the ROM. The total width for the ROM is thus 3x(7+7+9) = 69.

## Inner Encoder

The standard inner encoder is the same as the outer encoder, being a rate 1/2 four state systematic recursive convolutional encoder. As the interleaver outputs three consecutive bits in parallel, we form a rate 3/6 encoder. This is done similarly as for the outer encoder. From (3) and (4) and incrementing $i$ by 1, we have for the main data

$$\begin{aligned} y_{i+2}^0 &= x_{i+2}, \\ y_{i+2}^1 &= x_{i+2} + x_{i+1} + s_{i+1}^0 + s_{i+1}^1 \\ &= x_{i+2} + x_{i+1} + x_i + s_i^1, \\ s_{i+3}^0 &= x_{i+2} + x_{i+1} + s_{i+1}^1 = x_{i+2} + x_{i+1} + s_i^0, \\ s_{i+3}^1 &= x_{i+1} + s_{i+1}^0 + s_{i+1}^1 = x_{i+1} + x_i + s_i^1, \end{aligned} \tag{20}$$

and for the tail we have

$$\begin{aligned} y_{i+2}^0 &= s_{i+2}^0 + s_{i+2}^1 = 0, \\ y_{i+2}^1 &= s_{i+2}^1 = 0, \\ s_{i+3}^0 &= 0, \\ s_{i+3}^1 &= s_{i+2}^0 = 0. \end{aligned} \tag{21}$$

Letting $i = 3j$, we have $z_j^h = x_{i+h}$ as the input data, $w_j^h = y_{i+h}^0$ as the encoded data, $w_j^{3+h} = y_{i+h}^1$ as the encoded parity, for $0 \leq h \leq 2$, $s_j^0 = s_i^0$, $s_j^1 = s_i^1$, $s_{j+1}^0 = s_{i+3}^0$ and $s_{j+1}^1 = s_{i+3}^1$. This gives the equations for the rate 3/6 encoder for $0 \leq j \leq I/3–1$ as

$$\begin{aligned} w_j^0 &= z_j^0, \\ w_j^1 &= z_j^1, \\ w_j^2 &= z_j^2, \\ w_j^3 &= z_j^0 + s_j^0, \\ w_j^4 &= z_j^1 + z_j^0 + s_j^0 + s_j^1, \\ w_j^5 &= z_j^2 + z_j^1 + z_j^0 + s_j^1, \\ s_{j+1}^0 &= z_j^2 + z_j^1 + s_j^0, \\ s_{j+1}^1 &= z_j^1 + z_j^0 + s_j^1, \end{aligned} \tag{22}$$

and for the tail with $j = I/3$ as

$$\begin{aligned} w_j^0 &= s_j^0 + s_j^1, \\ w_j^1 &= s_j^0, \\ w_j^2 &= 0, \\ w_j^3 &= s_j^1, \\ w_j^4 &= s_j^0, \\ w_j^5 &= 0. \end{aligned} \tag{23}$$

## Parity Puncturing

Unlike data puncturing, parity puncturing is much more elegant and simpler to implement. Let $q_i$ be the puncturing value (either 0 or 1) for the parity bit from $0 \le i \le I–1$. That standard uses the variable $e_i$ (where we have added the index $i$), but to reduce complexity, we use the variable $f_i = e_i–1$. We have that

$$\begin{aligned} f_{i+1} &= (f_i - \Delta) \bmod I, \\ q_{i+1} &= u(f_i - \Delta), \end{aligned} \tag{24}$$

where $\Delta$ is the number of bits to be punctured from the $I$ parity bits, $f_0 = 0$, $q_0 = 1$ and

$$u(x) = \begin{cases} 1 \ ; \ x \ge 0, \\ 0 \ ; \ x < 0. \end{cases} \tag{25}$$

Note that the tail data and parity bits are not punctured. As we need to output three punctured bits at a time, we calculate the following values

$$\begin{aligned} m_h &= (h(I - \Delta) \bmod I) - I, \\ d_h &= (h(I - \Delta) \operatorname{div} I) \bmod 2. \end{aligned} \tag{26}$$

for $1 \le h \le 3$. We have

$$\begin{aligned} f_{i+h} &= (f_i + m_h) \bmod I, \\ c_{i+h} &= u(f_i + m_h). \end{aligned} \tag{27}$$

For $h = 1$, we have that $q_{i+1} = c_{i+1}$. However, for $q_{i+2}$ we need to also add $d_2$ (the number of overflows) and subtract $q_{i+1}$ (the number of previous overflows) modulo 2. A similar operation is also required for $q_{i+3}$. That is, we have

$$\begin{aligned} q_{i+1} &= c_{i+1}, \\ q_{i+2} &= (c_{i+2} + d_2 - q_{i+1}) \bmod 2 \\ &= c_{i+2} \oplus c_{i+1} \oplus d_2, \\ q_{i+3} &= (c_{i+3} + d_3 - q_{i+2} - q_{i+1}) \bmod 2 \\ &= c_{i+3} \oplus c_{i+2} \oplus d_3 \oplus d_2. \end{aligned} \tag{28}$$

That is, given $f_i$, $m_h$ for $1 \le h \le 3$, $d_2$ and $d_2 \oplus d_3$ stored in LUTs we can calculate the three following puncturing bits. We only need to store $f_i$+3 for the next calculation.

## Data and Parity Combining

The symbol interleaver is separated in two non–equal halves, with the first half containing the data and the second half containing the parity. Thus, the data and parity bits are separately combined after puncturing. That is, we combine the three data bits $z_j^h$, $0 \le h \le 2$, $0 \le j \le I/3$ with the corresponding three data puncturing bits $p_j^h$ to give a sequence that ranges from 0 to 3 bits. Similarly, we combine the three parity bits $w_j^h$ with the three parity puncturing bits $q_j^h = q_{3j+h}$ to give a sequence that ranges from 0 to 3 bits. As the tail is not punctured, we always write two bits for $z_{I/3}^h$ and $w_{I/3}^h$.

## Symbol Interleaver

The symbol interleaver first writes all the data bits, followed by all the parity bits, column by column into a memory with $m$ rows and 8100 columns, where the number of signal points is $2^m$, with $m$ = 2, 3, 4, 5, and 6 for QPSK, 8PSK, 16APSK, 32APSK and 64APSK, respectively. Data is then read row by row for each signal point.

However, as we need to write up to six bits at a time, we use six separate memories to store the data. The first three memories are used to store the data bits and the second three memories are used to store the parity bits.

Each RAM has a separate write address counter for one third of the number of rows (8100/3 = 2700) and another counter for the column. The number of columns is determined by how many columns are used by the data and parity bits. We have

$$\begin{aligned} N_d &= \lceil S/8100 \rceil, \\ N_p &= \lceil P/8100 \rceil. \end{aligned} \tag{29}$$

where $N_d$ and $N_p$ are the number of columns for $S$ data bits and $P$ parity bits, respectively. The maximum value of $S$ is 45429 (for ACM = 27) with $N_d$ = 6 and the maximum value of $P$ is 9234 (for ACM = 23) with $N_p$ = 2.

Thus, we use 12 bit counter for the row address, a 3 bit counter for the data column address and a 1 bit counter for the parity column address. Each counter is increment according to the number of bits being written (0 to 3) with the write enables to each RAM being appropriately selected. For example, if two data bits are being written with the previous last bit to be written to memory 1 at row $j$ and column $k$, then the first bit would be written to memory 2 at row ($j$+1) mod 2700 and column $k$+(($j$+1) div 2700). The second bit would be written to memory 0 at its corresponding row and column.

For the parity RAM, we need to also use $S_m = S \bmod 3$ to indicate the start of the write enable counter. $S_d = S$ div 3 and $S_m$ are also used to indicate the initial values of the row counters. For the first, second and third parity RAMs, the initial values are $S_d + u(S_m–1)$, $S_d + u(S_m–2)$ and $S_d$, respectively.

Ideally, each data RAM would be of size 2700x6 where the row address corresponds to one of the 2700 addresses and the column address to one of the 6 data bits. With a ping–pong memory, the size would be 5400x6. Each data

RAM is implemented using three 8Kx2 memories, with a 16Kx1 write and 8Kx2 read. Similarly, each of the parity RAMs would ideally be of size 5400x2, which is implemented using one 8Kx2 memory with a 16Kx1 write and 8Kx2 read.

To read $m$ bit symbol $s_i$, $0 \le i \le 8099$, we use the read address $i$ mod 2700 going to all six RAMs, reading a total of 8 bits (6 for data and 2 for parity) from the data and parity RAM $i$ div 2700. We use $u(i - S)$ to determine how we combine the 6 bit data and 2 bit parity. For example, with ACM = 27, if $i < S$, the output symbol is $(d_0,d_1,d_2,d_3,d_4,d_5)$ corresponding to the six bits selected from the data RAM. For $i \ge S$, the output is $(d_0,d_1,d_2,d_3,d_4,$ $p_0)$ where $(p_0,p_1)$ corresponds to the two bits selected from the parity RAMs.

## Frame Marker

A length 256 frame marker is attached to the beginning of each frame. This is used to allow a demodulator to synchronise to the phase and start of the encoded data. The marker is generated using an 8–bit Gold code and modulated using π/2–BPSK. The marker sequence is

$$s_i = z_i^0 + z_i^1 \tag{30}$$

where $0 \le i \le 255$, the addition is modulo 2 and

$$\begin{aligned} z_{i-8}^0 &= z_i^0 + z_{i-4}^0 + z_{i-5}^0 + z_{i-6}^0, \\ z_{i-8}^1 &= z_i^1 + z_{i-1}^1 + z_{i-3}^1 + z_{i-4}^1 + z_{i-5}^1 + z_{i-6}^1 \end{aligned} \tag{31}$$

are the two feedback equations for polynomials $g0(x) = 1+x^4+x^5+x^6+x^8$ and $g1(x) = 1+x^1+x^3+x^4+$ $x^5+x^6+x^8$, respectively. We have that $g0(x)$ is a primitive polynomial that generates a length 255 sequence and $g1(x)$ is an irreducible polynomial that can generate three different length 85 sequences. We define two 8–bit registers as

$$Z_i^j = \sum_{h=0}^{7} 2^h z_{i-h}^j \tag{32}$$

where $0 \le j \le 1$. We have that

$$\begin{aligned} z_i^j &= Z_i^j \bmod 2, \\ Z_{i+1}^j &= 128 z_{i-8}^j + (Z_i^j \text{ div } 2). \end{aligned} \tag{33}$$

where the initial values are $Z_0^0 = 150$ and $Z_0^1 = 73$.

## Frame Descriptor

A length 64 frame descriptor with π/2–BPSK modulation follows the frame marker. This uses a rate $R = 7/64$ orthogonal code with a minimum distance of $d_{\min} = 32$ that encodes five bits for the ACM value, one bit for PILOT, with one bit reserved for future upgrades. The asymptotic $E_b/N_0$ (energy per bit to single sided noise density ratio) coding gain is $10\log_{10}(2Rd_{\min}) = 8.45$ dB. We use a 6–bit counter to encode the descriptor where the counter value at time $i$, $0 \le i \le 63$ is

$$\begin{aligned} C_i &= \sum_{h=0}^{5} 2^h c_i^h, \\ C_{i+1} &= C_i + 1. \end{aligned} \tag{34}$$

The encoded sequence is

$$s_i = \left(a_6 + \sum_{i=0}^{5} c_i^h a_{5-h} + r_i\right) \bmod 2 \tag{35}$$

where $a_6$ = PILOT ($b_6$ in the standard), $a_4$ down to $a_0$ corresponds to ACM[4:0] ($b_1$ to $b_5$ in the standard), $a_5 = 0$ is the reserved bit ($b_7$ in the standard) and $r_i$ corresponds to a randomising sequence equal to E89B1C39AC244BF5 in hexadecimal (left most bit is $r_0$).

For each ACM value, Table 3 gives the values for Mod (modulation), $K$, $I$, $S_{\text{sur}}$, $S$, $\Delta$, $P$, $N = S+P$ (total number of encoded bits), $R_{\text{eff}} = K/8100$ (bandwidth efficiency, not including frame header and pilots) and SS (Signal Set, see Symbol Mapper section).

## Symbol Combiner

The symbol combiner is used to first select the π/2–BPSK modulation symbols used in the frame header (while FHR is high), the 6–bit coded symbols from the encoder (while SR is high) or the optional pilot symbols (while PR is high). As π/2–BPSK uses a subset of QPSK symbols (00 and 11 for even $i$ and 01 and 10 for odd $i$) we represent π/2–BPSK using two–bit QPSK symbols. For the pilot symbol, we represent this as a two bit QPSK symbol equal to 00.

The symbol combiner also shifts the 6–bit output so that the least significant bit as at the right most position. For example, for QPSK the symbol RAM output $(s_0,s_1,s_2,s_3,s_4,s_5)$ is output as S[5:0] = $[0,0,0,0,s_0,s_1]$.

## Symbol Mapper

The symbol mapper takes the 6 bit symbol S[5:0] and according to the ACM value, outputs the corresponding 12 bit I and Q values, using two's complement values. There are a total of 10 different signal sets used; one for QPSK, one for 8PSK, four for 16APSK, three for 32APSK and one for 64APSK. Using a single LUT, would require a memory of size 236x24. However, as all the signal sets are 90° rotationally symmetric, we can reduce the memory address space by one quarter to 59x22 (using 11 bits for the magnitude of I and Q).

For QPSK, 16APSK and 64APSK, the right most 0, 2 and four bits, respectively, are used to address the lookup table (adding an offset depen–

**Table 3: Encoder Constants**

| ACM | Mod | $K$ | $I$ | $S_{sur}$ | $S$ | $\Delta$ | $P$ | $N$ | $R_{eff}$ (bit/sym) | SS |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | QPSK | 5758 | 8640 | 300 | 8642 | 1084 | 7558 | 16200 | 0.71086 | 0 |
| 2 | QPSK | 6958 | 10440 | 300 | 10442 | 4684 | 5758 | 16200 | 0.85901 | 0 |
| 3 | QPSK | 8398 | 12600 | 274 | 11510 | 7912 | 4690 | 16200 | 1.03679 | 0 |
| 4 | QPSK | 9838 | 14760 | 251 | 12351 | 10913 | 3849 | 16200 | 1.21457 | 0 |
| 5 | QPSK | 11278 | 16920 | 234 | 13200 | 13922 | 3000 | 16200 | 1.39235 | 0 |
| 6 | QPSK | 13198 | 19800 | 218 | 14390 | 17992 | 1810 | 16200 | 1.62938 | 0 |
| 7 | 8PSK | 11278 | 16920 | 292 | 16470 | 9092 | 7830 | 24300 | 1.39235 | 1 |
| 8 | 8PSK | 13198 | 19800 | 240 | 15842 | 11344 | 8458 | 24300 | 1.62938 | 1 |
| 9 | 8PSK | 14878 | 22320 | 250 | 18602 | 16624 | 5698 | 24300 | 1.83679 | 1 |
| 10 | 8PSK | 17038 | 25560 | 234 | 19939 | 21201 | 4361 | 24300 | 2.10346 | 1 |
| 11 | 8PSK | 19198 | 28800 | 221 | 21218 | 25720 | 3082 | 24300 | 2.37012 | 1 |
| 12 | 8PSK | 21358 | 32040 | 214 | 22857 | 30599 | 1443 | 24300 | 2.63679 | 1 |
| 13 | 16APSK | 19198 | 28800 | 255 | 24482 | 20884 | 7918 | 32400 | 2.37012 | 2 |
| 14 | 16APSK | 21358 | 32040 | 241 | 25741 | 25383 | 6659 | 32400 | 2.63679 | 2 |
| 15 | 16APSK | 23518 | 35280 | 230 | 27051 | 29933 | 5349 | 32400 | 2.90346 | 3 |
| 16 | 16APSK | 25918 | 38880 | 220 | 28515 | 34997 | 3885 | 32400 | 3.19975 | 4 |
| 17 | 16APSK | 28318 | 42480 | 211 | 29880 | 39962 | 2520 | 32400 | 3.49605 | 5 |
| 18 | 32APSK | 25918 | 38880 | 245 | 31755 | 30137 | 8745 | 40500 | 3.19975 | 6 |
| 19 | 32APSK | 28318 | 42480 | 234 | 33137 | 35119 | 7363 | 40500 | 3.49605 | 6 |
| 20 | 32APSK | 30958 | 46440 | 224 | 34677 | 40619 | 5823 | 40500 | 3.82198 | 6 |
| 21 | 32APSK | 33358 | 50040 | 217 | 36197 | 45739 | 4303 | 40500 | 4.11827 | 7 |
| 22 | 32APSK | 35998 | 54000 | 210 | 37802 | 51304 | 2698 | 40500 | 4.44420 | 8 |
| 23 | 64APSK | 33358 | 50040 | 236 | 39366 | 40808 | 9234 | 48600 | 4.11827 | 9 |
| 24 | 64APSK | 35998 | 54000 | 228 | 41042 | 46444 | 7558 | 48600 | 4.44420 | 9 |
| 25 | 64APSK | 38638 | 57960 | 220 | 42507 | 51869 | 6093 | 48600 | 4.77012 | 9 |
| 26 | 64APSK | 41038 | 61560 | 214 | 43915 | 56877 | 4685 | 48600 | 5.06642 | 9 |
| 27 | 64APSK | 43678 | 65520 | 208 | 45429 | 62351 | 3171 | 48600 | 5.39235 | 9 |

dent on the ACM value), while the two most significant bits are used to perform a two's complement on the I and Q magnitude values if needed.

For 8PSK and 32APSK, as there are points on the I and Q axis, this complicates the address to the LUT and determining the signs of the I and Q values. A swap circuit for I and Q is also required.

The signal set is normalised so that the root mean square (RMS) value is equal to $\sqrt{2}\,1024$ = 1448.15. For example, QPSK has points at (1024,1024), (–1024,1024), (–1024,–1024) and (1024,–1024).

For 16APSK, 32APSK and 64APSK the signal points are distributed in rings with equidistant signal points in each ring. The ring ratio is defined as $\gamma_r = R_{r+1}/R_1$, where $R_r$, $1 \le r \le m-2$ is the ring radius from the inner ring with radius $R_1$ to the outer ring with radius $R_{m-2}$. Thus, 16APSK has two rings with $\gamma_1 = R_2/R_1$, 32APSK has three rings with $\gamma_1$ and $\gamma_2 = R_3/R_1$ and 64APSK has four rings with

$\gamma_1$, $\gamma_2$ and $\gamma_3 = R_4/R_1$. Table 4 gives the values for $\gamma_1$, $\gamma_2$ and $\gamma_3$ for the signal set indicated by SS.

**Table 4: Signal Sets**

| SS | Signal Set | $\gamma_1$ $\gamma_2$ $\gamma_3$ |
|---|---|---|
| 0 | QPSK | |
| 1 | 8PSK | |
| 2 | 16APSK | 3.15 |
| 3 | 16APSK | 2.85 |
| 4 | 16APSK | 2.75 |
| 5 | 16APSK | 2.60 |
| 6 | 32APSK | 2.84 5.27 |
| 7 | 32APSK | 2.72 4.87 |
| 8 | 32APSK | 2.54 4.33 |
| 9 | 64APSK | 2.73 4.52 6.31 |

## Pseudo Randomiser

The I and Q values of the coded symbols and pilot are pseudo randomised using an 18–bit Gold code. The randomiser outputs are defined by

$$\begin{aligned} r_i^0 &= z_i^0 + z_i^1, \\ r_i^1 &= z^0_{(i+2^{17}) \bmod 2^{18}-1} + z^1_{(i+2^{17}) \bmod 2^{18}-1} \\ &= z^0_{i-15} + z^0_{i-6} + z^0_{i-4} + z^1_{i-15} + z^1_{i-14} + z^1_{i-13} + \\ & z^1_{i-12} + z^1_{i-11} + z^1_{i-10} + z^1_{i-9} + z^1_{i-8} + z^1_{i-6} + z^1_{i-5} \end{aligned} \tag{36}$$

where $0 \le i \le 129599 + 3840a_6$ (16 codeword segments of 8100 symbols and 240 pilots each) and

$$\begin{aligned} z^0_{i-18} &= z_i^0 + z^0_{i-7}, \\ z^1_{i-18} &= z_i^1 + z^1_{i-5} + z^1_{i-7} + z^1_{i-10} \end{aligned} \tag{37}$$

are the two feedback equations for polynomials $g0(x) = 1+x^7+x^{18}$ and $g1(x) = 1+x^5+x^7+x^{10}+ x^{18}$, respectively. We define two 18–bit registers as

$$Z_i^j = \sum_{h=0}^{17} 2^h z^j_{i-h} \tag{38}$$

where $0 \le j \le 1$. We have that

$$\begin{aligned} z_i^j &= Z_i^j \bmod 2, \\ Z^j_{i+1} &= 2^{17} z^j_{i-18} + (Z_i^j \operatorname{div} 2). \end{aligned} \tag{39}$$

where the initial values are $Z_0^0$ = PRN[17:0] > 0 and $Z_0^1 = 2^{18}-1 = 262143$. The standard specifies a value $n$, $0 \le n \le 2^{18}-2$, that is used to determine PRN[17:0]. If we define $Z_0^0 = 1$, then PRN[17:0] = $Z_n^0$. For example, if $n = 1$, then PRN[17:0] = $2^{17}$.

We have that the output RN[1:0] = $[r_i^1, r_i^0]$. The $I$ and $Q$ outputs from the LUT are randomised according to Table 5. Randomisation is applied to all coded and pilot symbols, but not to the frame header symbols.

**Table 5: I and Q randomisation**

| RN[1:0] | I[11:0] | Q[11:0] |
|---|---|---|
| 0 | $I$ | $Q$ |
| 1 | $-Q$ | $I$ |
| 2 | $-I$ | $-Q$ |
| 3 | $Q$ | $-I$ |

## Encoding Operation

The input data X[1:0] = $X_i = [x_i^1, x_i^0] = [x_{2i+1}, x_{2i}]$, $0 \le i \le K/2-1$, where $i$ corresponds to XA[14:0], is input two bits at a time to the outer rate 2/3 convolutional encoder and into one half of the interleaver RAM. The other half of the interleaver memory has input data read into the inner rate 3/6 convolutional encoder in each CLK cycle.

The BUSY output indicates when the encoder can accept data. When high this indicates that new data must not be input to the encoder. That is, START must remain low while BUSY is high. If BUSY is low, the START signal is used to start the encoder by going high for one CLK cycle.

Figure 3 illustrates the encoder input timing. XR will go high one CLK cycle later after START goes high and will stay high for $K/2-1$ CLK cycles. The first data input must be input one CLK cycle after START and must be continuously input for a total of $K/2$ CLK cycles. A data address output XA[14:0] is provided for reading data from an external synchronous read input memory. Signal START and XR can ORed together to form the read enable.

When XA = $K/2-1$, XF will go high for one CLK cycle. BUSY will then go high for one CLK cycle while the tail bits are calculated. If the other half of the Input RAM is available, BUSY will go low, indicating that the next block may be input. If both halves of the RAM are full, then BUSY will stay high. BUSY will not go low again until one of the halves of the RAM becomes available. If BUSY is low, START can go high. To ensure a continuous output, data should be input as soon as possible after BUSY goes low. Figure 3 shows START going high again for the case where BUSY = 0.

Inputs X[1:0], START, ACM[4:0] and PILOT must be synchronous to CLK. Outputs XA[14:0], XR, XF and BUSY are synchronous to CLK. Internal encoding uses CLK.

The input data can be input in any ACM order. That is, it is not necessary to wait for the encoder to output the last block of one code before changing to another code. If changing the code, the encoder parameters ACM[4:0] and PILOT must stay constant from the time START goes high to until

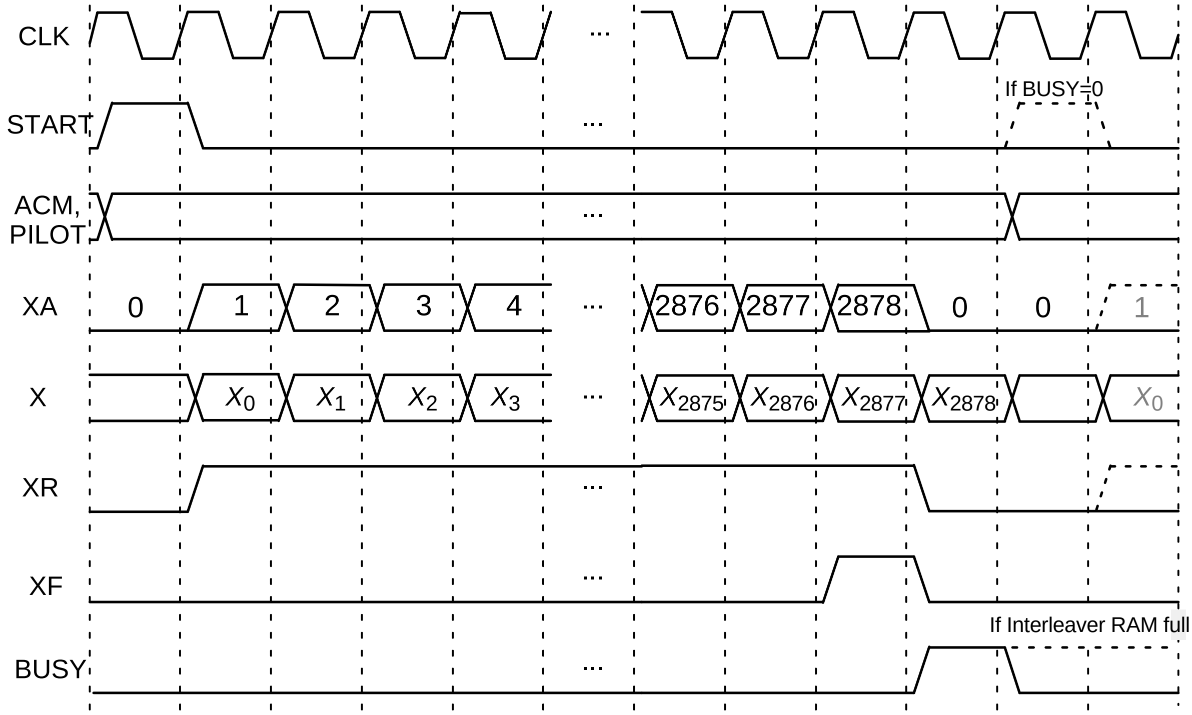


Figure 3: SCCC encoder input timing (ACM = 1).

one CLK cycle after XF goes high. As the frame header is only inserted every 16 codeword segments (CWS), ACM[4:0] and PILOT should only be changed after multiples of 16 CWS have been input.

Input PRN[17:0] is synchronous to SCLK and must not be changed during encoding. Outputs SI, SQ, S, RN, ACMO, FR, FHR, SR, PR and FF are synchronous to SCLK.

Figure 4 illustrates the encoder output timing. The frame ready signal FR goes high when data is ready to be output. The user can select either the symbol inphase SI[11:0] and symbol quadrature SQ[11:0] values, or the symbol S[5:0] values and scrambling sequence RN[1:0]. If frame header ready FHR or pilot ready PR are high then S[1:0] should be used to select a QPSK output. For FHR high, the QPSK points selected by S[1:0] will effectively produce a $\pi$/2–BPSK modulated signal. If symbol ready SR is high, the appropriate modulation as indicated by ACMO[4:0] should be selected. For SR and PR high, the $I$ and $Q$ values produced from S[5:0] should be scrambled according to Table 5.

The encoder will first produce a 256 symbol frame marker and 64 symbol frame descriptor. The encoder will then output 16 CWS with each CWS having 8100 code symbols plus 240 pilot symbols if PILOT = 1. The total number of symbols in each frame is $256+64+16(8100+240a_6) = 129920+3840a_6$.

If PILOT = 1, each CWS is subdivided into 15 subsections. Each subsection consists of 540 code symbols followed by 16 pilot symbols. Before scrambling, a pilot symbol is equal to point 00 of the QPSK signal set.

If data is input at an insufficient rate into the encoder, FR will go low after one and before 16 CWS. That is, the number of symbols output will be $320+c(8100+240a_6)$, where $1 \leq c \leq 16$. Once sufficient data has been received, a new frame will be output, begining with the frame marker.

## Encoder Speed

To ensure that a continuous output is formed, The total time to input the data must not exceed the time to output an encoded stream. The input time is limited by the inner encoder and is equal to $(K/2+8)T_c+2T_s$ where $T_c$ and $T_s$ are the clock periods of CLK and SCLK, respectively.

The extra clock cycles consist of one CLK cycle for encoding the tail for the outer encoder, two CLK cycles to read the inner encoder interleaver parameters, two CLK cycles to read the data from the interleaver RAM, one CLK cycle for encoding the tail for the inner encoder, two SCLK cycles to start outputting the encoded symbols (going from the CLK to the SCLK domain) and two CLK cycles to indicate that inner encoding has completed (going from the SCLK to the CLK domain).

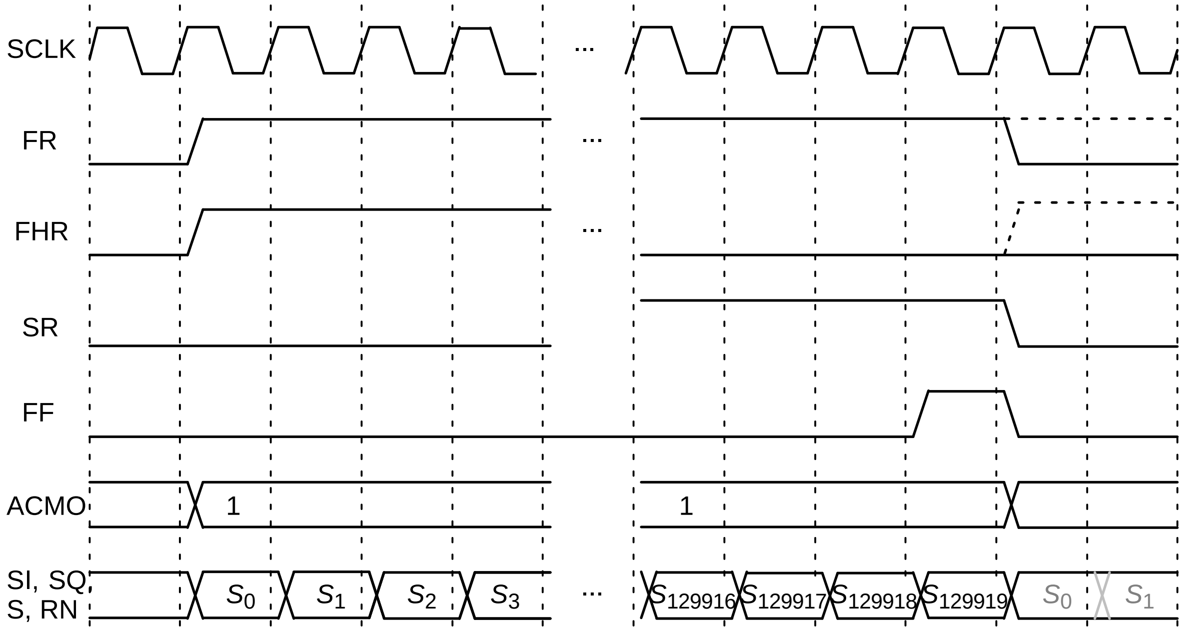


Figure 4: SCCC encoder output timing (ACM = 1, PILOT = 0).

When sending signals from the CLK to SCLK domain and vice–versa, this must done only from FF to FF, with no logic in between, otherwise glitches from the LUT outputs can cause incorrect operation. Thus dual FF falling edge detectors are used to indicate the end of inner encoding.

The minimum output time is equal to $(8100+240a_6)T_S$ where $a_6$ = PILOT. Thus, we have

$$(K/2+8)T_c+2T_s \leq (8100+240a_6)T_s. \quad (40)$$

Thus, if given $f_S = 1/T_S$ (SCLK frequency), the minimum $f_C = 1/T_C$ (CLK frequency) is given by

$$f_c \geq \frac{(K/2+8)f_s}{8098\,+\,240a_6} = F(a_6)f_s. \quad (41)$$

Table 6 gives the value of the clock scaling factor $F(a_6)$ for $K$ and PILOT. If $F \geq 1$, the maximum encoding speed is determined by CLK and we have

$$f_e = \frac{K}{(K/2+8)T_c+2T_s} = \frac{Kf_c}{K/2+8+2F(a_6)}. \quad (42)$$

If $F < 1$ the maximum encoding speed is determined by SCLK and we have

$$f_e = \frac{Kf_s}{8100\,+\,240a_6}. \quad (43)$$

Table 6 also gives $G(a_6) = f_e/f_C$ and $H(a_6) = f_e/f_S$. The factor $G$ is useful for determining the maximum encoding speed when CLK limits the encoding speed. The factor $H$ is useful when a fixed SCLK is used for all coding schemes.

For example, if $f_S$ = 100 MHz, we require $f_C \geq F(0)f_S$ = 269.783 MHz with data rates ranging from 69.041 Mbit/s (ACM = 1, PILOT = 1) to 539.235 Mbit/s (ACM = 27, PILOT = 0).

### Encoder Delay

The total encoder delay can be separated into two parts. This is the outer encoder delay $T_i$ and inner encoder delay $T_O$. Each delay is equal to

$$\begin{aligned} T_i &= (K/2\,+\,2)T_c, \\ T_o &= (K/2\,+\,6)T_c\,+\,5T_s. \end{aligned} \quad (44)$$

The total encoder delay is then

$$T_e = (K\,+\,8)T_c\,+\,5T_s. \quad (45)$$

Depending on the phase between CLK and SCLK, the actual encoder delay will vary from $T_e$–$T_S$ to $T_e$.

## Ordering Information

SW–SCE02C–SOP (SignOnce Project License)
SW–SCE02C–SOS (SignOnce Site License)
SW–SCE02C–VHD (VHDL ASIC License)

All licenses include EDIF and VHDL cores. The SignOnce and ASIC licenses allows unlimited instantiations. The EDIF core can be used for Virtex–2, Spartan–3 and Virtex–4 with Foundation or ISE software. The VHDL core can be used for Virtex–5, Spartan–6, Virtex–6, 7–Series, UltraScale and UltraScale+ with ISE or Vivado software.

Note that *Small World Communications* only provides software and does not provide the actual devices themselves. Please contact *Small World Communications* for a quote.

**Table 6: Clock scaling factor *F*(PILOT) = $f_c/f_s$, *G*(PILOT) = $f_e/f_c$ and *H*(PILOT) = $f_e/f_s$.**

| ACM | *K* | *F*(0) | *F*(1) | *G*(0) | *G*(1) | *H*(0) | *H*(1) |
|---|---|---|---|---|---|---|---|
| 1 | 5758 | 0.35651 | 0.34625 | 1.99421 | 1.99422 | 0.71086 | 0.69041 |
| 2 | 6958 | 0.43060 | 0.41821 | 1.99517 | 1.99517 | 0.85901 | 0.83429 |
| 3 | 8398 | 0.51951 | 0.50456 | 1.99595 | 1.99596 | 1.03679 | 1.00695 |
| 4 | 9838 | 0.60842 | 0.59091 | 1.99651 | 1.99651 | 1.21457 | 1.17962 |
| 5,7 | 11278 | 0.69733 | 0.67726 | 1.99692 | 1.99693 | 1.39235 | 1.35228 |
| 6,8 | 13198 | 0.81588 | 0.79240 | 1.99733 | 1.99734 | 1.62938 | 1.58249 |
| 9 | 14878 | 0.91961 | 0.89314 | 1.99760 | 1.99761 | 1.83679 | 1.78393 |
| 10 | 17038 | 1.05298 | 1.02267 | 1.99788 | 1.99788 | 2.10346 | 2.04293 |
| 11,13 | 19198 | 1.18634 | 1.15219 | 1.99809 | 1.99809 | 2.37012 | 2.30192 |
| 12,14 | 21358 | 1.31971 | 1.28172 | 1.99826 | 1.99826 | 2.63679 | 2.56091 |
| 15 | 23518 | 1.45307 | 1.41125 | 1.99839 | 1.99840 | 2.90346 | 2.81990 |
| 16,18 | 25918 | 1.60126 | 1.55517 | 1.99852 | 1.99853 | 3.19975 | 3.10767 |
| 17,19 | 28318 | 1.74944 | 1.69909 | 1.99862 | 1.99863 | 3.49605 | 3.39544 |
| 20 | 30958 | 1.91245 | 1.85740 | 1.99872 | 1.99873 | 3.82198 | 3.71199 |
| 21,23 | 33358 | 2.06063 | 2.00132 | 1.99879 | 1.99880 | 4.11827 | 3.99976 |
| 22,24 | 35998 | 2.22364 | 2.15963 | 1.99886 | 1.99887 | 4.44420 | 4.31631 |
| 25 | 38638 | 2.38664 | 2.31794 | 1.99893 | 1.99893 | 4.77012 | 4.63285 |
| 26 | 41038 | 2.53482 | 2.46186 | 1.99897 | 1.99898 | 5.06642 | 4.92062 |
| 27 | 43678 | 2.69783 | 2.62017 | 1.99902 | 1.99903 | 5.39235 | 5.23717 |

## References

[1] Consultative Committee for Space Data Systems, “Recommendation for space data system standards: Flexible advanced coding and modulation scheme for high rate telemetry applications,” CCSDS 131.2–B–1, Blue Book, Mar. 2012.



*Small World Communications*, 6 First Avenue,
Payneham South SA 5070, Australia.
info@sworld.com.au ph. +61 8 8332 0319
http://www.sworld.com.au fax +61 8 7117 1416

## Revision History

- 1.00 28 Dec. 2023. First release.