Chapter 2
Edge-Coded Signaling Techniques

When one door closes another door opens; but we so often look so long and so regretfully upon the closed door, that we do not see the ones which open for us.

Alexander Graham Bell

The objective of this chapter is to present the analysis of signaling protocols for data transfer over a single-wire achieving high data rates (in the Mbs range), low power consumption, and small footprint. The protocols do not require a CDR, can operate with signals at low amplitude voltage (∼1 V), has simple encoding and decoding schemes, and can tolerate baud rate differences between transmitter and receiver. We collectively refer to this new family as Edge-Coded Signaling (ECS) because its core idea is to transfer the indices of only the ON bits as a series of transition edges rather than bit times. A very compact packet header gives information about the number of such indices and the encoding operations to which the raw bits have been subjected. When the pulses are received, the receiver applies the appropriate decoding to infer the original data bits. The ECS protocols are dynamic in that they can accommodate several data rates. It exploits edge detection of incoming pulses to achieve remarkable robustness with respect to jitters, skews, and clock inaccuracies between the transmitter and the receiver. The protocols achieve significant improvements in data rate, reliability, packet security, and power efficiency with respect to state-of-the-art CDR-less techniques. ECS is also architecturally flexible in that it can be configured according to the signaling topology (Master–Slave, Ring, Star, etc.).

S. Muzaffar, I. M. Elfadel, *Secure, Low-Power IoT Communication Using Edge-Coded Signaling*, https://doi.org/10.1007/978-3-030-95914-2_2

2.1 Edge-Coded Signaling (ECS)

2.1.1 Edge-Coding Scheme

The core idea of ECS is to select the ON bits in a data word and transmit their index numbers as pulse streams instead of transmitting the data bits themselves. An example is given in Fig. 2.1 where the bit sequence "0101" (a) is transformed into series of pulses (b) in which the count of pulses in each series is $n + 1$ with n being the ordinal number of the ON bit in the binary sequence. In the example of Fig. 2.1b, there are two series of pulses. The first series has one pulse corresponding to the leading ON bit at position 0, and the second series has three pulses corresponding to the ON bit at position 2. One series of pulses is separated from an adjacent one by an inter-symbol separator α. Please note that α is not a time delay but rather a spacing or separation symbol that is measured in clock cycles with the clock-cycle count given by the local transmitter clock at transmission and the local receiver clock at reception. The clocks at both ends do not have to be synchronized. Also, note that one is always added to the pulse count corresponding to the index number. This

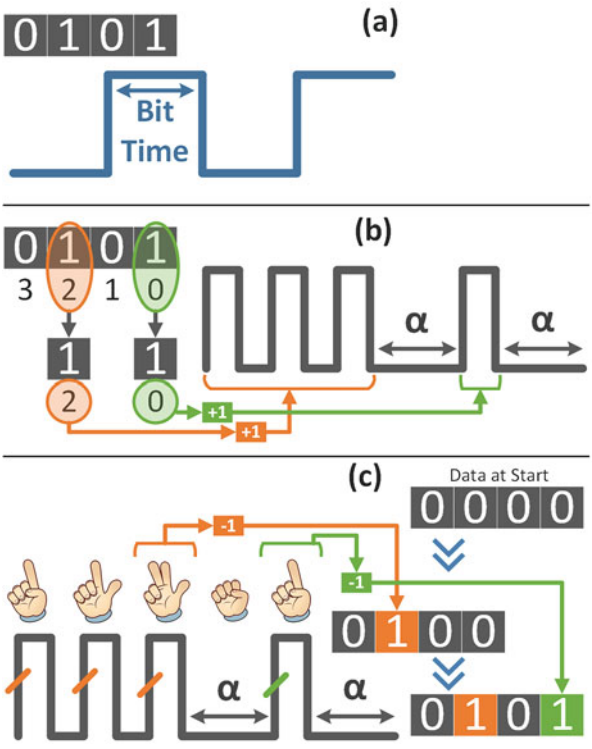

Fig. 2.1 (**a**) Standard serial transfer. (**b**) Edge-coded transmitter. (**c**) Edge-coded receiver

operation is necessary to handle the transmission of index 0. Otherwise, no pulse will be transferred if the bit at index 0 is ON. For each input pulse series, the ECS receiver counts the number of the incoming rising edges, subtracts one to retrieve the index number (i.e., $n = PulseCount - 1$), and sets a data bit at the index number. This is shown in Fig. 2.1c. The apparent drawback is that more work is seemingly needed to transmit such pulse series than the raw bits themselves. However, this is not the case as it is conceivable to achieve high data rates, using an encoding process that makes the index numbers as small as possible. This is accomplished by breaking the bit stream into smaller segments, reducing the number of ON bits as much as possible in each segment and relocating these ON bits to the lowest index positions. The encoding information and the number of ON bits in the encoded data are sent as a packet header along with the index numbers. All the information in the packet header itself is transmitted as pulse streams, exactly as the index numbers. In short, instead of transmitting bits, ECS codes them as edge counts and transmits them along with the formatting information, itself edge-coded, so that the receiver is able to reconstitute the data word. The steps involved in ECS transmission are explained in the following subsections.

2.1.2 ECS Segmentation

The number of pulses to transmit increases rapidly with the data word size B and the number of its ON bits. The most significant bits require larger number of pulses to represent their index numbers. Considering the worst case where all the bits are ON, the number of pulses required would be $B(B+1)/2$. The rapid increase in the number of pulses reduces the data rate rapidly and, therefore, the count of pulses must be limited. To do so, ECS breaks the data word into smaller segments of size $l = 4$ bits each, thus limiting the index numbers to a maximum of 3 (i.e., $4 + 3 + 2 + 1$). With the reduced segment size, the maximum number of pulses per segment reduces to 10. An ECS segmentation example is given in Fig. 2.2 where a 16-bit word is partitioned into 4-bit segments $S_i, 1 < i < 4$. The segmentation steps are also shown in Algorithm 1 on lines 1 and 2. The optimization of the segment size l is discussed in Sect. 2.2.

2.1.3 ECS Encoding

The increase in data word size also increases the number of inter-symbol separators needed to separate out the pulse streams, representing the ON bits. Such separators reduce the data rate significantly. Reducing the number of ON bits helps in mitigating the effect of separators on data rate. ECS encoding effectively reduces the number of ON bits in each data segment. The ECS encoding is simply a conditional bit-wise NOT operation on a target segment, with the condition being that the

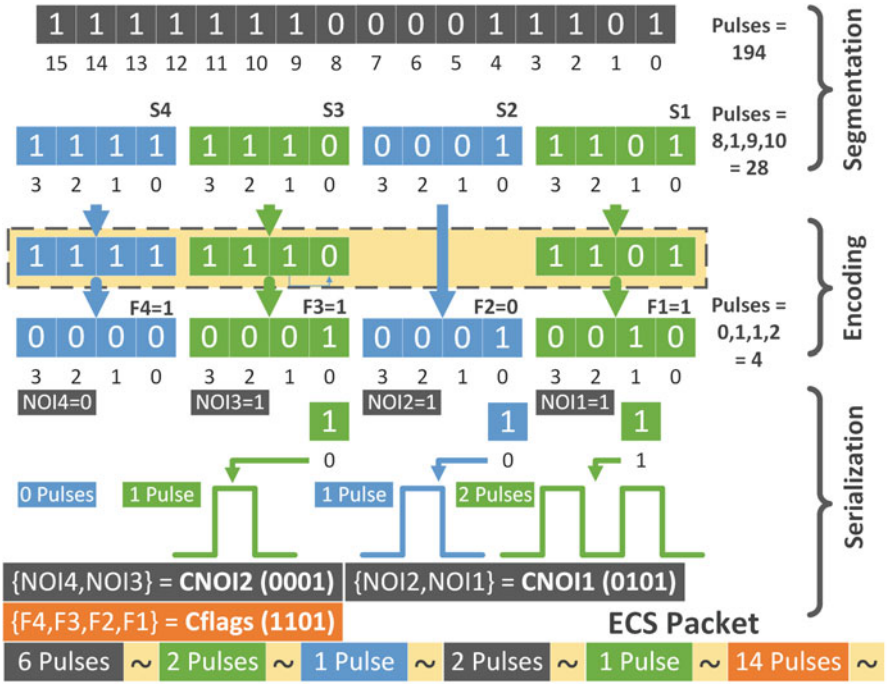

Fig. 2.2 Example: ECS packet formation

Algorithm 1 ECS segmentation and encoding process

Inputs:
- Data: 16-bit data word

Outputs:
- $CFlags$: Concatenated Encoding Indicators
- $CNOI_1, CNOI_2$: Concatenated ON-bit Counts

1: $S_1 = Data[3:0]$, $S_2 = Data[7:4]$
2: $S_3 = Data[11:8]$, $S_4 = Data[15:12]$
3: **for** i=1 **to** 4 **do**
4: $NOI_i = countONbits(S_i)$
5: $F_i = 0$
6: **if** $NOI_i > l/2$ **then**
7: $S_i = \sim S_i$
8: $F_i = 1$
9: $NOI_i = countONbits(S_i)$
10: **end if**
11: **end for**
12: $CFlags = \{F_4, F_3, F_2, F_1\}$
13: $CNOI_1 = \{NOI_2, NOI_1\}, CNOI_2 = \{NOI_4, NOI_3\}$

number of ON bits in a segment is longer than half of the segment size. To explain the encoding scheme further, let us assume $B = 16$. If a segment satisfies the said condition, bits of the segment are inverted and a 1-bit flag, F_i, is set to represent the applied operation. The subscript i represents the segment number. The encoding steps are presented in Algorithm 1 from lines 3–11 and in the encoding section of Fig. 2.2. Each segment is processed independently, and four distinct flags are generated, one for each segment. These four flags are then concatenated to yield a single 4-bit flag named $CFlags$. Additionally, the encoding process produces four 2-bit *Numbers of Indices*, each, denoted NOI_i, representing the number of ON bits in segment i. The $NOIs$ of two consecutive segments are concatenated to yield two 4-bit *concatenated* $NOIs$ that are denoted $CNOIs$. The generation of $CFlags$ and $CNOIs$ is shown in Algorithm 1 on lines 12 and 13 and in the serialization section of Fig. 2.2. At the end of this process, all the information required for transmission gets compacted in nibbles of 4 bits each, which is the same as the size of the 4-bit segment and, hence, helps in maximizing the data rate. The efficient hardware implementation of an encoder performing segmentation and encoding is discussed in Sect. 2.4.

2.1.4 ECS Transmitter

Pulse Stream and Separator Generation Scheme

In the ECS transmission process, the ECS scheme for generating the pulse streams and the inter-symbol separators plays a crucial role. The encoding pulses in the ECS packet and the α spacings between packets are generated using the ECS clock, which can be obtained in two ways. One way is for the system clock to be routed directly to the ECS clock port. Another way is for the system clock to be divided to generate a slower ECS clock. The pulse generation process is illustrated in Fig. 2.3. The ECS clock is ANDed with a control signal, *Pulse Stream Active* (PSA), set by the control module. The PSA is high during the transmission of a pulse stream, allowing the ECS clock cycles to go through. During the transmission of the inter-symbol separator α, PSA is low, thus gating the ECS clock. Please note that α in ECS is not a time delay, but rather a count of the rising or falling edges of the ESC clock. In Fig. 2.3, we have used $\alpha = 4$ clock cycles as it is the optimal count at which the maximum data rate is achieved. This will be discussed further in Sect. 2.2.

Transmission Flow

The format of the ECS packet is shown in Fig. 2.4 and a numerical example is given in Fig. 2.2. The $CFlags$ are transmitted to inform the receiver about the encoding process, while the $CNOIs$ are transmitted to help the receiver account for all the incoming ON bit indices. The ECS transceiver starts the transmission by sending a

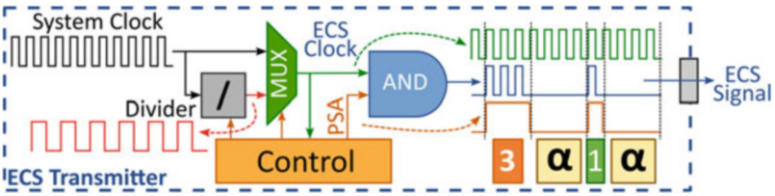

Fig. 2.3 ECS pulse stream and inter-symbol separator generator (indices from Fig. 2.1)

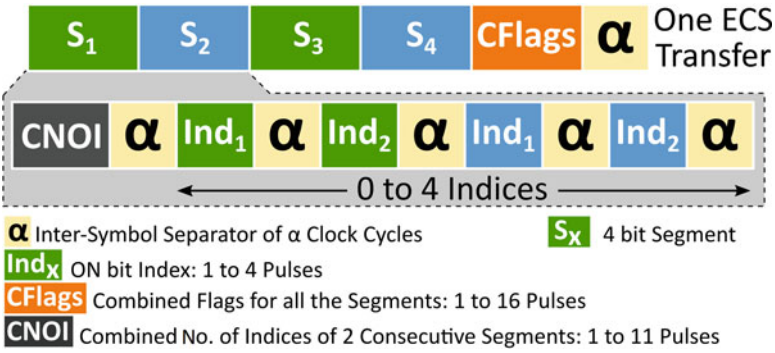

Fig. 2.4 ECS packet

pulse stream with a pulse count equal to $CNOI_1 + 1$ followed by an inter-symbol separator α of four clock cycles (line 2 in Algorithm 2). The additional pulse is needed to inform the receiver not to expect any index number when the count of ON bits in the first two segments is zero. Next, the transmitter sends a number of pulse streams equal in count to $NOI_1 + NOI_2$ followed by an α at the end of each stream. The pulse count in each pulse stream is equal to the index number of an ON bit in segments S_1 and S_2 plus an additional pulse, making a total of $index + 1$ pulses. The additional pulse is used to handle the transmission of a zero index number. The transmission process of indices is presented in Algorithm 2 on lines 3 through 9. A similar transmission follows for the next two segments, S_3 and S_4, during which $CNOI_2$ and the index numbers of the ON bits in these segments are transmitted. At the end of the transmission of all segments, the CFlags are transmitted, also in the form of a pulse stream followed by an α (line 10 in Algorithm 2). Their pulse count is equal to $CFlags + 1$. An additional pulse is needed to represent zero content of $CFlags$ as in the case when no segment goes through the encoding inversion. The graphical transmission process and the generated waveforms are shown in Fig. 2.6.

Algorithm 2 ECS transmitter algorithm

Inputs:
- $CFlags$: Concatenated Encoding Indicators
- $CNOI_1$, $CNOI_2$: Concatenated ON-bit Counts **Outputs:**
- ECS Signal: The pulse streams and inter-symbol separators

```
for j=1,2 do
    sendPulsesWithSeparator(CNOI_j + 1, α)
    for each ON bit in S_{2j-1} with index i do
        sendPulsesWithSeparator(i + 1, α)
    end for
    for each ON bit in S_{2j} with index i do
        sendPulsesWithSeparator(i + 1, α)
    end for
end for
sendPulsesWithSeparator(CFlags + 1, α)
```

2.1.5 *ECS Receiver*

Pulse Stream and Separator Reception

The ECS receiver is unique in that it does not require any clock and data recovery (CDR) circuitry either to receive the incoming data over single channel or to synchronize it with a local clock. The ECS exploits detection and count of edges of the incoming pulse streams to receive all the information required to rebuild the transmitted data successfully. Contrary to standard serial transfer, the width of transmitted pulses is inconsequential to ECS and, hence, does not employ this information in receiving data. Though there could be different implementations of the ECS receiver, our implementation is focused on counting the number of clock cycles between two ECS pulses to detect the inter-symbol spacing and separate out the incoming pulse streams. The ECS pulse stream reception process is illustrated in Fig. 2.5. The ECS receiver keeps track of two counts, the pulse count and the clock count. The reception process starts with the very first rising edge of the input pulse stream. At each rising edge of the pulse stream, the pulse count is incremented, and

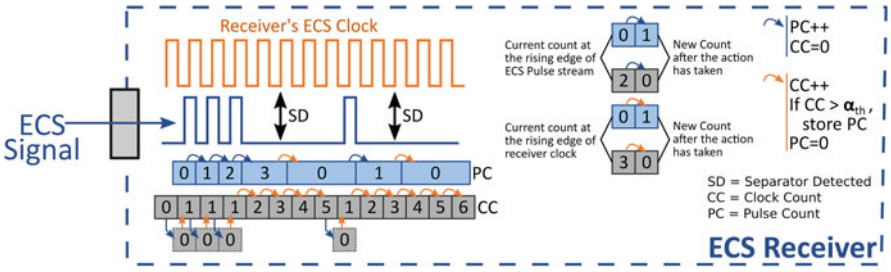

Fig. 2.5 ECS pulse stream and inter-symbol separator receiver (input signal from Fig. 2.3)

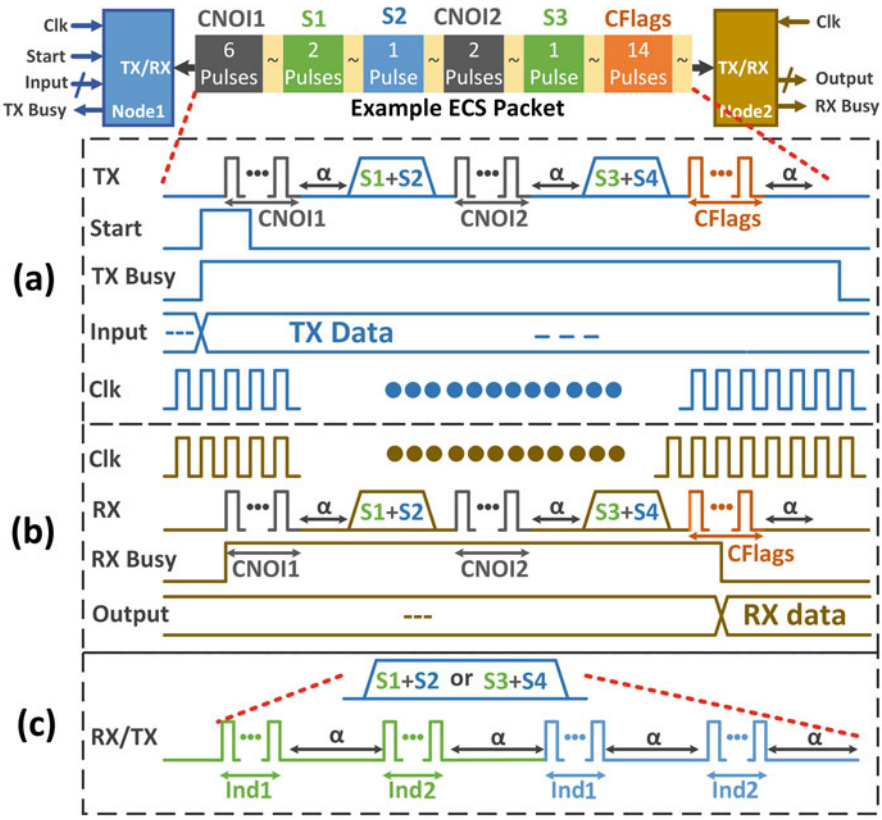

Fig. 2.6 (**a**) Transmitter. (**b**) Receiver. (**c**) Indices

the clock count is cleared. At each rising edge of the receiver's ECS clock, the clock count is incremented and compared with a separator threshold α_{th} that is set to half of α in our implementation of the protocol. If the condition is satisfied, an inter-symbol separator is declared, and the current pulse count is accordingly stored as a record of the transmitted packet.

Reception Flow, Decoding, and Reconstruction

The ECS packet reception process starts with the very first rising edge of the first pulse stream for $CNOI_1$. The pulse stream is received as described in the previous subsection. Similarly, all the following pulse streams for the indices of S_1, S_2, and $CNOI_2$ and the indices of S_3, S_4, and $CFlags$ are received, and the corresponding parts of the ECS packet are updated. Bits of each segment are complemented if the corresponding bit in $CFlags$ is set. At this stage, all the transmitted segments are assembled to rebuild the data word. The full process of receiving, decoding, and

assembling is shown in Algorithm 3. An efficient implementation of ECS encoding and decoding is discussed in Sect. 2.4. The transmission and reception processes along with their generated waveforms are shown in Fig. 2.6.

Algorithm 3 ECS receiver algorithm

Inputs:
- ECS Signal: The pulse streams and inter-symbol separators

Outputs:
- Data: 16-bit data word

for i=1 **to** 2 **do**
  $CNOI_i$ =PulseStreamReceiver()*−1
  $NOI = CNOI_i[1:0]$
  $S_{2i-1} = S_{2i} = 0$
  for j=1 **to** NOI **do**
    $index$ =PulseStreamReceiver()−1
    $S_{2i-1}[index] = 1$
  end for
  $NOI = CNOI_i[3:2]$
  for j=1 **to** NOI **do**
    $index$ =PulseStreamReceiver()−1
    $S_{2i}[index] = 1$
  end for
end for
$CFlags$ =PulseStreamReceiver()−1
$Data = \{S_4 \oplus \{4\{CFlags[3]\}\}, S_3 \oplus \{4\{CFlags[2]\}\}, S_2 \oplus \{4\{CFlags[1]\}\}, S_1 \oplus \{4\{CFlags[0]\}\}\}$

* PulseStreamReceiver() is the pulse counter for each input pulse stream (Fig. 2.5)

2.1.6 ECS Transmission System

The ECS communication technique can be used with a variety of channels such as wired, wireless, infrared, and human body channel. ECS is advantageous to all these channels as it results in significant simplification of transceiver circuitry, reduction in power consumption, and decrease in footprint. A simple PHY layer for single-wire communication is shown in Fig. 2.7a where two tri-state buffers are used to switch channel access between transmitter and receiver. Moreover, ECS can be used with any communication medium without any significant change, as shown in Fig. 2.7b. In case of wireless transmission, the wireless front end can be easily used to transmit and receive the packet pulses. As ECS does not need power-hungry circuits such as CDR or duty cycle correction, the complexity of the front end reduces significantly as compared to standard transceivers. Additionally, ECS helps in improving the bit rate of wireless transmission. For example, the standard OOK and ASK modulation techniques need duty cycle accuracy to recover square pulses. As the transmission data rate increases, the output pulses turn into triangular

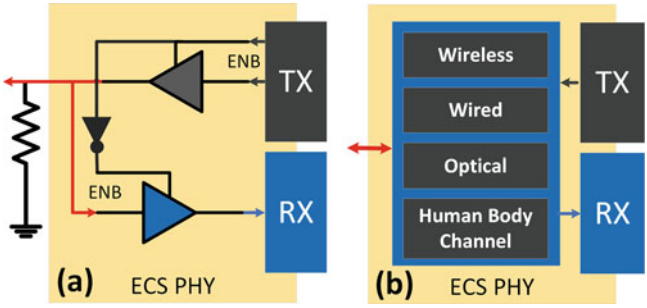

Fig. 2.7 (**a**) ECS PHY for single-wire. (**b**) General ECS PHY block diagram

ones at the receiver end of the wireless modules, which limits the bit rate. On the other hand, ECS does not depend on the duty cycle and can correctly decode the triangular and demodulated pulses as long as their peaks remain above the detection threshold of the ECS receiver. This is also the case with infrared channels where the power consumption and complexity of the optoelectronic system are reduced. ECS can also help in reducing the transceiver complexity of human body-channel communication (BCC) [69] by eliminating duty cycle dependency and the need for CDR while enabling the processing of pulses deformed through the variable-gain human body channel. ECS has been successfully tested with all these channels, and the experiments are discussed in Sect. 2.4.

2.1.7 ECS Data Rate

Let b_i^s be the i-th bit in the s-th encoded segment and l be the number of bits per segment. The total number of segments, N, is given by

$$N = \frac{B}{l} \tag{2.1}$$

In the ECS packet, the $CNOIs$, $CFlags$, and segments all have the same length l. Therefore, each $CFlag$ represents l consecutive segments among a total of N segments. The number of $CFlags$, n_{cf}, in the ECS packet is given by

$$n_{cf} = \frac{N}{l} \tag{2.2}$$

Similarly, each $CNOI$ concatenates the $NOIs$ of 2 consecutive segments among a total of N segments. The number of $CNOIs$, n_{cn}, in the ECS packet is therefore given by

$$n_{cn} = \frac{N}{2} \tag{2.3}$$

For a segment s, the required number of pulses is given by

$$P_s = \sum_{i=0}^{l-1} (i+1) b_i^s \tag{2.4}$$

and the number of ON bit indices is given by

$$NOI_s = \sum_{i=0}^{l-1} b_i^s \tag{2.5}$$

Let PI_x be the number of pulses required for one $CNOI$. The PI_x is given as

$$PI_x = 1 + NOI_{2x-1} + 2^{l/2} NOI_{2x} \quad , \quad 1 \le x \le n_{cn} \tag{2.6}$$

where one additional pulse is used to represent the absence of ON bits. The x subscript in (2.6) refers to two consecutive segment numbers, one odd and one even, for NOI_s in (2.5). Now, let PF_y be the number of pulses required for one $CFlags$. PF_y is given as

$$PF_y = 1 + \sum_{i=0}^{l-1} 2^i F_s \quad , \quad s = i + l(y-1) \quad , \quad 1 \le y \le n_{cf} \tag{2.7}$$

where F_s is the flag bit for the s-th encoded segment. Again, one additional pulse is used to represent the *no-encoding* state.

The number of pulses for $CNOIs$, $Segments$, and $CFlags$ and the total number of NOI pulses are, respectively, given by

$$n_{pi} = \sum_{x=1}^{n_{cn}} PI_x \tag{2.8}$$

$$n_{ps} = \sum_{s=1}^{N} P_s \tag{2.9}$$

$$n_{pf} = \sum_{y=1}^{n_{cf}} PF_y \tag{2.10}$$

$$n_{in} = \sum_{s=1}^{N} NOI_s \tag{2.11}$$

The total pulse count is therefore given by

$$C = \left(n_{cf} + n_{cn} + n_{in}\right)\alpha + n_{pi} + n_{ps} + n_{pf} \tag{2.12}$$

where α is the number of clock cycles for an inter-symbol separator. The data rate R of the ECS protocol for a bit stream of B bits, clock period of T, and a total pulse count of C is given as

$$R = \frac{B}{TC} \tag{2.13}$$

The optimum values of the protocol parameters are derived in Sect. 2.2.

2.2 ECS Optimizations

2.2.1 Optimum Inter-symbol Separator α

Since all the transmitted pulse streams are separated by an inter-symbol separator, an appropriate value of α is indispensable for successful packet reception and for maximizing data rate. Keeping all the parameters in (2.12) and (2.13) constant except for α, we obtain the relationship $R \propto a/(b + c\alpha)$, where a, b, and c are constants. This relationship clearly shows that an increase in α decreases data rate, as shown in Fig. 2.8a. Both empirically and theoretically [49], the smallest value of α for guaranteeing correct decoding is 4 clock cycles [49]. Below this value, the receiver would fail to decode the packet successfully because of the ambiguity between pulse spacing and inter-symbol separators. A value of α larger than 4 will increase the tolerance to clock variations but decrease data rate and reduce reliability with respect to packet failure.

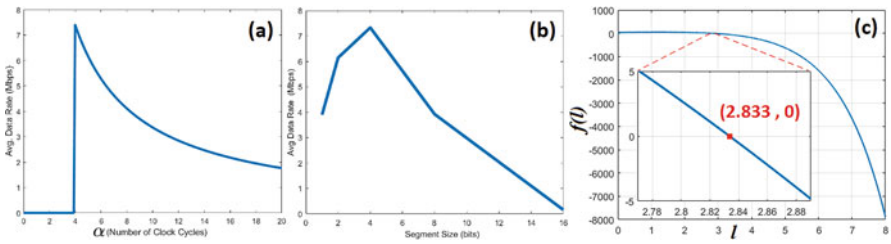

Fig. 2.8 (**a**) Data rate vs. α. (**b**) Data rate vs. segment size. (**c**) $f(l)$ vs. segment size l (Eq. (2.18))

2.2.2 Optimum Segment Length l

To find the optimum segment length that maximizes data rate, we minimize the number of clock cycles needed to transmit the ECS packet. We know from the previous section that the number of segments is $N = B/l$. Assuming that bits 0 and 1 are equally likely, the expected value of P_s in (2.4) is

$$E[P_s] = \frac{l(l+1)}{4} \tag{2.14}$$

Similarly, the expected value of PI_x in (2.6) is

$$E[PI_x] = \frac{2 + l(1 + 2^{l/2})}{2} \tag{2.15}$$

Assuming the flag bit F_s is equally likely to be 0 or 1, the expected value of PF_y in (2.7) is

$$E[PF_y] = \frac{1 + 2^l}{2} \tag{2.16}$$

Using $N = B/l$, the expected value of the total number of clock cycles C as given in (2.12) becomes

$$E[C] = \left(\frac{B}{l^2} + \frac{B}{2l} + \frac{B}{2}\right)\alpha + \frac{B}{2l} + \frac{B(1 + 2^{l/2})}{2} + \frac{B(l+1)}{4} + \frac{B(1 + 2^l)}{2l^2} \tag{2.17}$$

Taking the derivative with respect to l and equating it with zero (i.e., $\partial E[C]/\partial l = 0$), we get

$$\begin{aligned} f(l) \triangleq \frac{\partial E[C]}{\partial l} &= \alpha(8 + 2l) + 2l - l^3(2^{l/2}\ln(2) + 1) \\ &\quad -2^l(2l\ln(2) - 4) + 4 \\ &= 0 \end{aligned} \tag{2.18}$$

A graphical method to find the optimal segment length l_{opt} for a given α is to plot $f(l)$ as function of l and find the l intercept point. Such plot is shown in Fig. 2.8c for $\alpha = 4$, which results in

$$l_{opt} = 2.833 \approx 3 \quad bits \tag{2.19}$$

ECS divides the data word into segments of equal size and, therefore for a word size that is a power of 2, there are two possibilities of optimum segment length, 2 and 4. We select $l_{opt} = 4$ because there is a negligible reduction in data rate as compared

to $l = 2$ where the degradation is significant. If the segment length is increased or decreased from the optimum value of 4, the data rate degrades rapidly, as shown in Fig. 2.8b. Segments smaller than 4 bits reduce the data rate due to the increased number of inter-symbol separators. In contrast, segment lengths larger than 4 bits affect the data rate negatively due to the increase in most significant bit (MSB) index numbers. To achieve maximum data rate, the protocol must be operated with segments of length 4 bits each.

2.3 Earlier Versions of ECS

ECS1 and ECS2 are earlier versions of the ECS3 protocol described in the previous sections. With slight differences, these techniques apply an encoding scheme to a data word B to *minimize* the number of ON bits and *move* them to the least significant bit (LSB) end of the packet with the goal of lowering the number of pulses required to transmit the data bits. The encoding process includes a segmentation step where the data is broken into N independent segments of size l bits each (i.e., $N = B/l$). To maximize data rate, they use, on each segment, an encoding combination of bit inversion and/or segment reversion/flipping. For ECS1, this combination is meant to reduce the number of ON bits and decrease their index values. For ECS2, the same combination is intended to reduce the number of ON bits and decrease the decimal number represented by each segment. To facilitate decoding, flag pulses representing the type of encoding performed are added to each segment. Unlike ECS1, the ECS2 segment flags of two consecutive segments are combined in one data word flag and placed in the header. ECS2 further applies a third segmentation step post-encoding, the level-2 segmentation, whose goal is to further reduce the number of pulses per segment and, therefore, increase the data rate.

All the pieces of information including flags, the number of indices, and the indices themselves in the case of ECS1, or the decimal numbers of each segment in the case of ECS2, are transmitted in the form of pulse streams. The pulse is characterized by its width which is the number of clock cycles during which it remains high. Within a given packet, segment pulse streams are separated by an inter-symbol separator α. The ECS1 and ECS2 packet formats are presented in Figs. 2.9 and 2.10. To describe the process of ECS1 and ECS2 data transmission, examples are given in Figs. 2.11 and 2.12, respectively. A decimal number 65,055 is considered as a 16-bit data word for transmission. The 16-bit data word is divided into two independent segments, each of 8 bits, which reduces the index numbers of MSB bits and, consequently, the number of pulses to represent the ON bits. Because the number of ON bits in Segment#1 is higher than half of the segment length (5 and 4, respectively), the bits are inverted, and the Flags of Segment#1 are set to 2. This step further reduces the number of ON bits in Segment#1, but the index numbers of the ON bits are located in the MSB part of the segment. The bit-wise flip operation is therefore applied to relocate the ON bits to the LSB part, which results in the

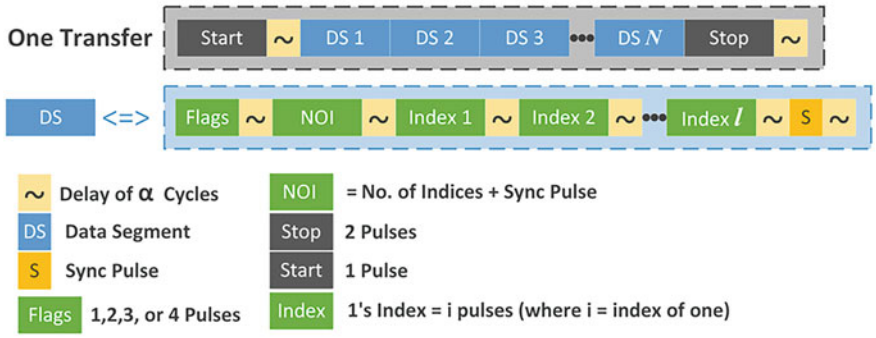

Fig. 2.9 ECS1 packet format

Fig. 2.10 ECS2 packet format

reduction of the number of pulses. The $Flags$ field of Segment#1 is now updated to 3, signifying that both of the encoding operations are applied. The same steps are applied to Segment#2 except that only the inversion operation is needed. The $Flags$ field of Segment#2 is set to 2, signifying that only the inversion operation is applied. In the case of ECS1, all the packet information including the encoded segments, $Flags$, and the number of ON bit locations ($NOIs$) is now available to start the transmission. However, in the case of ECS2, an additional segmentation step is applied where each encoded segment is divided into two sub-segments. All the pieces of information are transmitted in the form of pulse streams separated by inter-symbol separators.

The receiver counts the number of pulses for each pulse stream and applies the decoding according to the $Flags$ field in the received packet. Like ECS3, ECS1 has a variable number of symbols per data word, which enables the addition of security layers, whereas ECS2 presents a fixed number of symbols per data word, which improves transmission reliability with respect to packet failures. The three protocols of the ECS family are compared in Table 2.1 for a 16-bit data word transmission.

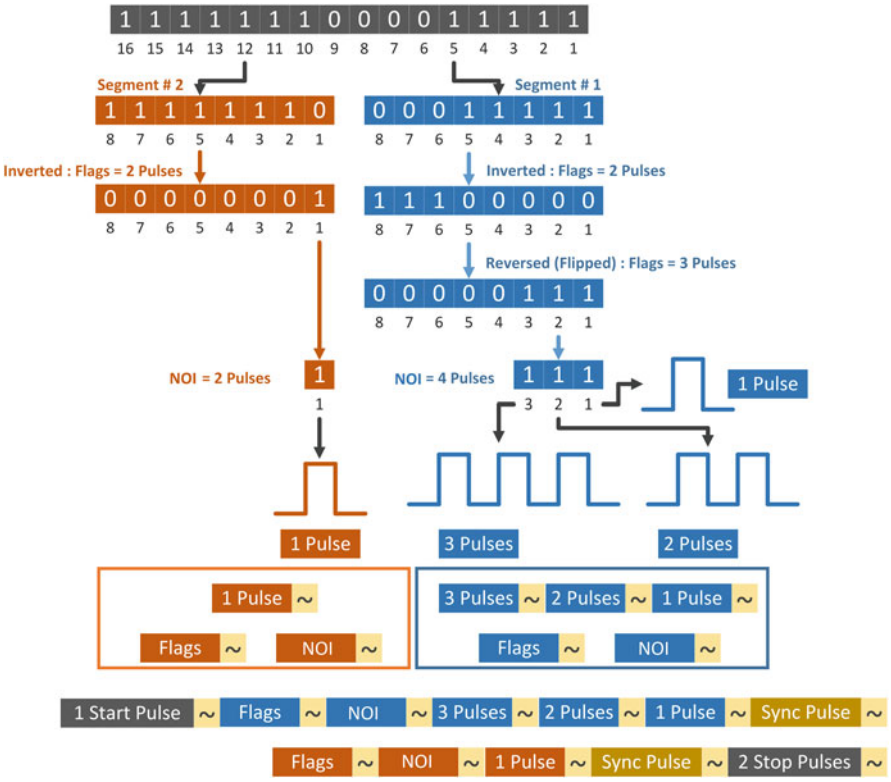

Fig. 2.11 ECS1 encoding and packetization example

2.3.1 Data Rates

ECS1 and ECS2 share with ECS3 the same notation and data rate equations as given in Sect. 2.1.7. However, the mathematical definitions of some of the symbols may vary as per the differences in their packet structure. All these variations are presented in Table 2.2. The generalized data rate equations for the ECS family are shown in Rows 14 and 15. Therein, the symbol n_{pe} represents configuration pulses that include start, stop, and sync pulses.

2.3.2 Optimizations

The segment size is chosen to maximize data rate. For a small segment, the inter-symbol separators inserted between pulse streams to separate symbols reduce the data rate. Similarly for large segments, ON bits with high indices require a large

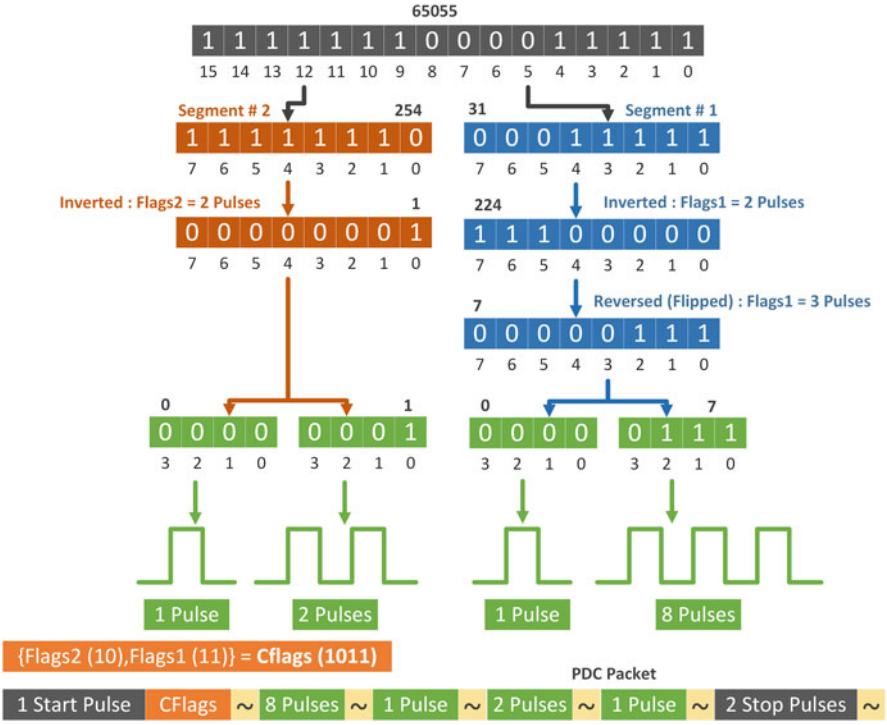

Fig. 2.12 ECS2 encoding and packetization example

number of pulses to be transmitted, which in turn reduces the data rate. It is therefore intuitive that there is a segment size for which the data rate is maximum. For ECS1, the data rate is maximized when the number of bits per segment is 8. For ECS2, the level-2 sub-segment (SS) size is appropriately chosen to maximize the data rate. The number of pulses increases exponentially with the size of SS, which in turn drastically reduces the data rate. For ECS2, the data rate is maximized when the number of bits per sub-segment is 4. The process of finding the optimum segment size is the same as the one presented in Sect. 2.2.2.

2.4 Experimental Setups and Results

An ECS communication system is implemented in Verilog HDL over Xilinx Spartan-6 FPGA board and verified through simulation and real-time communication between two nodes. For an apple-to-apple comparison with the earlier versions of the protocol, similar communication systems are developed for ECS1 and ECS2. However, this section describes only the ECS implementation. The development of the ECS system includes the efficient hardware implementation of ECS encoder

Table 2.1 Comparison of ECS family member techniques using 16-bit data word

		ECS1	ECS2	ECS3
Packetization				
Segment size	Level 1	8	8	4
	Level 2	–	4	–
No. of segments	Level 1	2	2	4
	Level 2	–	4	–
CFlags	Count	2	1	1
	Size (bits)	2	4	4
CNOIs	Count	2	–	2
	Size (bits)	2	–	4
Encoding steps		Invert + Flip	Invert + Flip	Invert
No. of symbols/packet		Dynamic	Fixed	Dynamic
Performance (25 MHz clock, 65nm CMOS technology)				
Data Rate	(Mb/s)	3.1–8.5 (4.1)[a]	4.8–12.9 (7.3)	4.2–26.7 (6.4)
	(% ↑)[d]		54.8–51.7 (78)[a]	35.5–214 (56)
Power	(μW)	≈26.6	≈25	≈19
	(% ↓)[d]		6	28.5
Eb	(pJ/bit)	3.1–8.5 (6.5)[a]	1.9–5.1 (3.4)	0.7–4.5 (2.9)
	(% ↓)[d]		38.7–40 (47.7)[a]	47–77.4 (55.4)
Area	(*Gatecount*)	≈2356	≈2150	≈2098
	(% ↓)[d]		8.7	10.9
Security[b]		Yes	No	Yes
Reliability	(NVLs)[c]	10–18	6	5–12
	(% ↑)[d]		up to 66.7	up to 72.2

[a] (Avg.)
[b] Packet protection
[c] No. of vulnerable locations
[d] Improvement relative to ECS1

and decoder. Both the encoder and decoder are combinatorial in nature and present a low power operation without any extra computational overhead. The encoder is implemented as a single hardware block that works with one 4-bit segment as input and generates the corresponding NOI, F_s, and the encoded segment. The encoder truth table is shown in Table 2.3, where S is the input data segment and S_E is the encoded segment at the output. Due to the segmentation and encoding process, there is a maximum of two ON bits per segment for which the index numbers Ind_1 and Ind_2 need to be transmitted. The ECS decoder at the receiver end takes as input all the received index numbers for a given segment and outputs a 4-bit data segment.

Table 2.2 Data rates of ECS member techniques

	Parameter	Notation	ECS1	ECS2	ECS3
1	Segment size		l	$l_1, l_2, l = l_2$	l
2	No. of segments	N	B/l	$N_{l_1} = B/l_1$ $N = N_{l_2} = \frac{N_{l_1}}{l_2} = \frac{B}{l_1 l_2}$	B/l
3	No. of $CFlags$	n_{cf}	N	$N_{l_1}/2$	N/l
4	No. of $CNOIs$	n_{cn}	N	0	$N/2$
5	Pulses/segment	P_s	$\sum_{i=0}^{l-1}(i+1)b_i^s$		
6	On bits/segment	NOI_s	$\sum_{i=0}^{l-1} b_i^s$	0	$\sum_{i=0}^{l-1} b_i^s$
7	Pulses/CNOI	PI_x^{a}	$1 + NOI_x$	0	$1 + NOI_{2x-1}$ $+ 2^{l/2} NOI_{2x}$
8	Pulses/CFlags	PF_y^{b}	$1 + f_0^y + 2f_1^y$	$1 + f_0^y + 2f_1^y$ $+ 4f_2^y + 8f_3^y$	$1 + \sum_{i=0}^{l-1} 2^i F_s$ $s = i + l(y-1)$
9	Total CNOIs pulses	n_{pi}	$\sum_{x=1}^{n_{cn}} PI_x$		
10	Total segment pulses	n_{ps}	$\sum_{s=1}^{N} P_s$		
11	Total CFlags pulses	n_{pf}	$\sum_{y=1}^{n_{cf}} PF_y$		
12	Total ON bits	n_{in}	$\sum_{s=1}^{N} NOI_s$	N	$\sum_{s=1}^{N} NOI_s$
13	Total extra pulses	n_{pe}^{c}	$3 + 2\alpha + N(2+\alpha)$	$3 + 2\alpha$	0
14	Total pulse count	C	$\left(n_{cf} + n_{cn} + n_{in}\right)\alpha + n_{pi} + n_{ps} + n_{pf} + n_{pe}$		
15	Data rate	R	$B \,/\, TC$		

[a] $1 \le x \le n_{cn}$
[b] $1 \le y \le n_{cf}$
[c] Start/stop and sync pulses

Table 2.3 ECS encoder

S	S_E	F	NOI	Ind_2	Ind_1
0000/1111	0000	0/1	00	000	000
0001/1110	0001	0/1	01	000	001
0010/1101	0010	0/1	01	000	010
0011	S	0	10	010	001
0100/1011	0100	0/1	01	000	011
0101/0110	S	0	10	011	001/010
0111/1000	1000	1/0	01	000	100
1001/1010/1100	S	0	10	100	001/010/011

The equations of the ECS decoder logic are

$$\{C, B, A\} = Ind_1 \tag{2.20}$$

$$\{F, E, D\} = Ind_2 \tag{2.21}$$

$$DS_1 = \{\bar{C}, B \cdot A, B \cdot \bar{A}, \bar{B} \cdot A\} \tag{2.22}$$

$$DS_2 = \{\bar{F}, E \cdot D, E \cdot \bar{D}, \bar{E} \cdot D\} \tag{2.23}$$

$$S_s = (DS_2 | DS_1) \oplus \{F_s, F_s, F_s, F_s\} \tag{2.24}$$

where DS_0 and DS_1 denote intermediate Verilog wires.

The ECS experimental setup comprises two nodes, each of which employs the abovementioned encoder and decoder along with the transmitter and receiver algorithms, respectively, all implemented in Verilog HDL. The ECS Transmission Algorithm 2 and Reception Algorithm 3 are synthesized as finite state machines. We have used 16-bit data words at a clock rate of 25 MHz. The transmitter at the first node sends a 16-bit data starting at 0 with an increment of 1 at each transmission. The second node resends the same data back. The returned and original data words are compared to verify the complete round-trip chain, and the number of perfect matches is logged. The ECS technique is verified using a number of single-channel links such as single-wire, wireless, infrared, and human body channel. In wired communication, a single-wire is used to connect both nodes using the PHY layer shown in Fig. 2.7. For wireless communication, a 433 MHz transceiver is used which accepts a raw ECS bit stream and transmits it wirelessly using OOK/ASK modulation. Similarly, for infrared communication, a simple infrared transceiver driver circuitry is used. For human body channel communication, new transceivers have been developed to carry out transmission through the human body. In all four experiments, ECS achieves flawless transmission. It must be noted that to support the aforementioned communication channels, ECS remains unchanged and only the front ends are replaced to transfer pulse streams through the desired channel. As highlighted in Sect. 2.1, ECS improves the bit rate of wireless transmission, which is verified with the experimental setup under discussion. Indeed we have observed an increase in bit rate from 4.8 to 20 Kbps (≈300% improvement). Similar observations

have been recorded while testing the infrared communication channel. For body channel communication, ECS helps in reducing the transceiver complexity, which is a significant advantage. In this latter case, a cascade of an amplifier, a discrete filter, and a level detector is sufficient to recover the ECS pulses traveling through the human body [54].

Along with the FPGA prototype, we have also synthesized and verified the ECS3 design using a Synopsis logic-synthesis flow and a GLOBALFOUNDRIES 65 nm process in order to get the most realistic area and power estimates and compare them with the published literature. We have determined that ECS3 consumes 19 μW with a gate count of 2098 gates, offering dynamic data rates in the range of 4.2–26.7 Mb/s (averaging 6.4 Mb/s) with a 25-MHz clock. As mentioned above, the selection of the 25 MHz clock rate is just for illustration purposes. We have verified the functionality of ECS3 using frequencies in the range from few KHz up to 200 MHz, the maximum frequency supported by our FPGA platform. As is clear from (2.13), the ECS3 data rate increases linearly with the clock frequency, and higher clock frequencies can be used to achieve higher data rates. Compared with NRZ serial transfer (NST) using CDR, ECS3 reduces area and power by more than 87% and 78%, respectively. Table 2.1 summarizes and compares the performance parameters of ECS1, ECS2, and ECS3. The data rate of ECS3 is increased significantly as compared to ECS1 and is as good as ECS2. ECS3 consumes less power and is more energy-efficient than ECS1 and ECS2 while maintaining a small footprint. Additionally, ECS3 helps in providing packet security, as will be discussed in Chap. 7. The reliability of ECS3 is similar to ECS2. It provides an improvement of up to 72.2% as compared to ECS1. In a nutshell, the results show that, overall, ECS3 outperforms both ECS1 and ECS2. Table 2.4 compares ECS3 with NST, which includes CDR, in terms of area and power. The main reason for the significant ECS3 advantage in area and power is that NST needs CDR to recover data successfully while ECS3 does not. CDR is the main source of power consumption, and even if we use the recently published low-power CDRs proposed in [12, 37, 38, 77], and [73], ECS3 still outperforms NST.

Table 2.4 ECS comparison with simple serial

	Power (μW)			Area (*Gatecount*)			
	SRL[a]	CDR	Total[d] (PI)[e]	SRL	CDR[c]	Total[d] (PI)[e]	
ECS3	19	N/A	19	2098	N/A	2098	65 nm
NST[b]	32.1	70	102.1 (81%)	1327	15,600	16,927 (87%)	90 nm [37]
		62.5	94.6 (80%)		60,000	61,327 (96%)	90 nm [38]
		90	122.1 (84%)		N/A	N/A	90 nm [12]
		57.5	89.6 (79%)		19,800	21,127 (90%)	65 nm [77]
		60.6	92.7 (80%)		N/A	N/A	28 nm [73]

[a] Serializer
[b] NRZ serial transfer
[c] Estimated calculation
[d] SRL+CDR
[e] %Increase as compared to ECS3

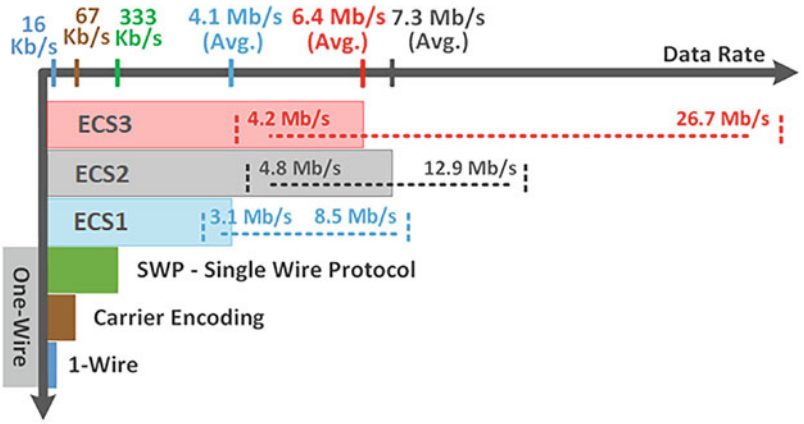

Fig. 2.13 Data rate consumption: one-wire protocols ([16, 27, 44]) vs. ECS1, ECS2, and ECS3

Furthermore, in Fig. 2.13, the ECS3 data rate is compared with the data rates of few existing CDR-less single-wire transmission techniques [16, 27, 44] as well as with ECS1 and ECS2. The comparison shows that ECS3 achieves significantly higher performance without any CDR and with tolerance toward jitters, skew, and clock inaccuracies. For small footprint applications (wireless sensor nodes, wearable computing, body-area networks, etc.) ECS3 is definitely the more reasonable choice.

2.5 Analysis

In this section, we further discuss the major characteristics of the ECS family based on the analytical, numerical, and experimental results we have obtained so far. The detailed timing and robustness analysis is provided in Chap. 3.

2.5.1 Data Rate

ECS is dynamic in that the actual data rate of the protocol is dictated by the pulse count which is very much data dependent. The statistical distributions of the ECS1, ECS2, and ECS3 data rates are shown in Fig. 2.14 for which exhaustive sampling of 16-bit data words is used with a total of $2^{16} - 1$ pseudo-random bit stream (PRBS). Each word is segmented and encoded as per the protocol specifications. For data rate calculations, we use a 25 MHz clock. Please note that the data rates are determined using both numerical simulations and hardware experiments. Comparing the histogram of ECS3 with that of ECS1, an increase in data rate is observed, ranging from 4.2 Mbps (35% ↑) to 26.7 Mbps (214% ↑) with an average

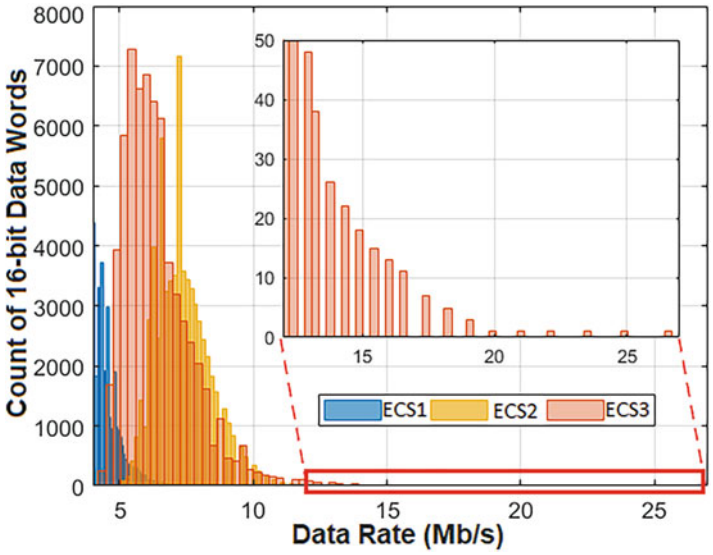

Fig. 2.14 Data rate histograms at 25 MHz clock

of 6.4 Mbps (56% ↑). Additionally, comparing the histogram of ECS3 with that of ECS2, it is observed that ECS3 outperforms ECS2 by achieving a maximum data rate of 26.7 Mbps (107% ↑).

2.5.2 Data Word Length and Complexity

As described in detail in Sect. 2.2.2, the optimum segment length for ECS to maximize data rate is 4 bits. Additionally, in Sect. 2.1, the encoding process generates one flag bit for each of the 4-bit long segments. The four flag bits are then concatenated to form another segment of optimum length (4 bits) that is known as the $CFlags$. Similarly, the process concatenates the $NOIs$ of two consecutive data segments to generate two additional segments of optimum lengths (4 bits each) that are known as the $CNOIs$. By the end of the encoding process, everything is packed in segments of optimum length. Because exactly four 4-bit data segments from the data word are needed to achieve this optimum configuration, the optimum data word size is 16 bits. We note that changing the length of ECS segments to incorporate data words of length larger than 16 bits has the potential of significantly decreasing the data rate while noticeably increasing the complexity, hardware resources, and power consumption.

In this context, ECS3 has a distinct advantage with respect to ECS1 and ECS2 when multiple words are being transmitted. To explain this advantage, we need to take a closer look at two hardware implementations: word-based and block-based.

Word-Based Implementation

The system is strictly designed to handle one word at a time. The transmission starts with setting the busy signal high, as shown in Fig. 2.6, and ends with clearing it when the transmission of a 16-bit word is complete. Let us denote the memory required to store the data word, $CFlags$, and $NOIs$ by M_W, M_F, and M_N bits, respectively. A word-based implementation needs a total of $1 \times M_W + 1 \times M_F + 1 \times M_N$ bits and only one pass to transmit a word, as shown in Fig. 2.15a. On the other hand, transmitting n_W 16-bit words in a word-based implementation incurs an additional delay of one clock cycle for each word to set up the input data port before starting transmission, which results in a total number of $n_W \times (1 + C)$ clock cycles where C is given in Eq. (2.12).

Block-Based Implementation

An alternative system design is to transmit the word block by breaking it into n_W 16-bit words and transmitting each word in a separate pass while keeping a busy signal high unless the transmission of all the words is complete. This is accomplished if the word-based implementation, described above, is used iteratively n_W times with an additional delay of one clock cycle at each pass to set up the transmission of next word in the block. The latency remains $n_W \times (1 + C)$ clock cycles as in the word-based implementation, and the throughput of ECS is unchanged. However, there is an increase in the complexity of the hardware as $n_W \times M_W$ bits of memory are needed to store the full word block. The memories M_F and M_N to store $CFlags$ and $NOIs$ remain unchanged. Additionally, control logic is needed to load the input data port with another 16-bit word from the block and to re-trigger the transmission process, a B_W-to-16 MUX to select a 16-bit word, and an adder to update the selection port of the MUX using a very simple pass control logic. The block diagram of such a block-based implementation is shown in Fig. 2.15b. The added memory and control logic will contribute to a slightly increased consumption of power and hardware resources with the throughput remaining unchanged.

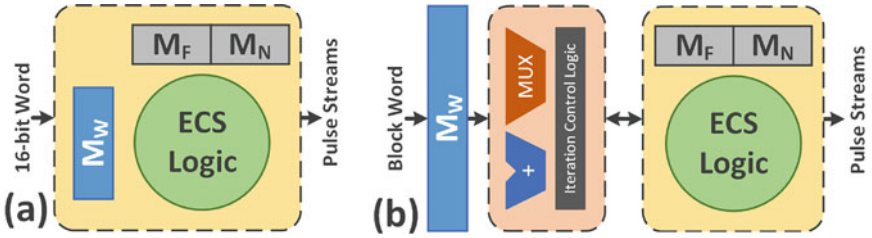

Fig. 2.15 (**a**) Word-based implementation. (**b**) Block-based implementation

2.5.3 *Error Detection and Correction*

During ECS transmission, there could be two main sources of transmission errors. These two sources are due to channel noise and are the following:

1. Packet Failure: Either one of the inter-symbol separators or the $CNOIs$ is garbled. As shown in Fig. 2.16a-i and a-ii, the receiver *fails to receive* the packet successfully because it expects a number of pulse streams that is different from what is actually transmitted.

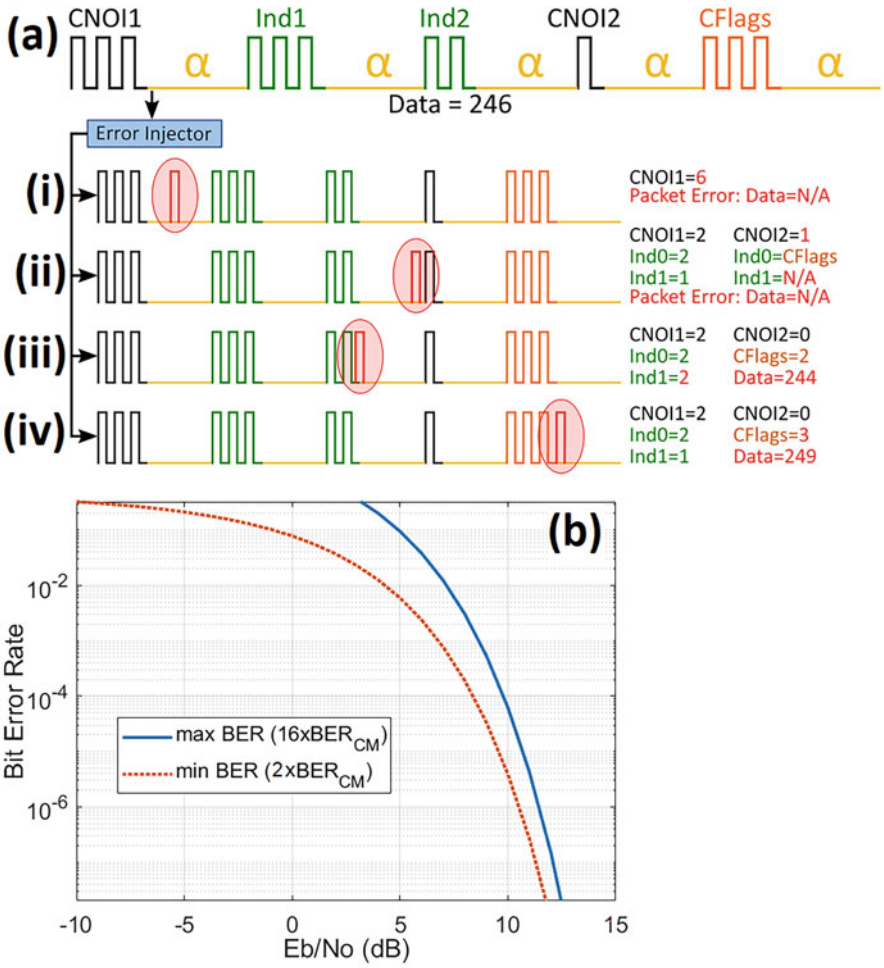

Fig. 2.16 (**a**) Errors in ECS packet reception (marked in red). (**b**) ECS BER using BPSK as modulation scheme

2. Data Corruption: The indices or the $CFlags$ pulse streams get garbled due to the addition or removal of pulses. In this scenario, as shown in Fig. 2.16a-iii, the packet *is received successfully*, but the index numbers are wrong, which results in data word errors at decoding. A similar scenario occurs if the $CFlags$ pulses are received with error, as shown in Fig. 2.16a-iv.

In our experimental setup, we have implemented an error injector that randomly inserts extra pulses into the ECS packets, as shown in Fig. 2.16a. The first type of error in the above list can be detected by monitoring the inter-symbol separator. In our implementation of the ECS protocol, if the separator is prolonged for more than twice α (i.e., $separator > 2\alpha$), the receiver declares packet failure, resets, and sends a request to retransmit the packet. As for the second type of error, it is the same as in a standard serial transfer, where one or more bits are in error. In our implementation, these bit errors are handled using a 1-bit parity code. However, there are several state-of-the-art error detection and correction techniques that ECS can use seamlessly as it does not prevent the preprocessing of data prior to encoding and transmission. Exploring such techniques and their compatibility with ECS is the subject of future work.

2.5.4 Bit Error Rate

Like data rate, the bit error rate (BER) of ECS is also dynamic and depends on the transmitted data. The BER of ECS depends on the BER of the PHY layer which may be using a standard modulation scheme [66] such as OOK, ASK, FSK, or BPSK, for transmitting the bit stream. The conceptual block diagram of such a setup is shown in Fig. 2.7b where ECS amounts to an encoding step prior to modulation. The bit stream, in the case of PHY with a modulation, is replaced with the pulse stream as generated by ECS. As a result, the BER of ECS is largely determined by the type of channel and the modulation used. ECS can help in reducing the complexity of the PHY front end by allowing them to focus on the amplitude of the recovered digital signals rather than other factors such as phase or bit width. In case there is a one-bit error in a unit time due to channel modulation, there would be one pulse in error for ECS that could affect the ECS packet in four different locations: within inter-symbol separator, within $CNOI$, within $CFlags$, or within an index number. These four different cases are shown in Fig. 2.16a and will impact the ECS BER differently according to the following rules:

$$BER_{ECS} = \begin{cases} 16 \times BER_{CM} \text{ if } b_e \in \alpha \\ 16 \times BER_{CM} \text{ if } b_e \in CNOI \\ 4 \times BER_{CM} \text{ if } b_e \in CFlags \\ 2 \times BER_{CM} \text{ if } b_e \in index \end{cases} \tag{2.25}$$

where b_e is the bit in error, BER_{ECS} is the ECS BER, and BER_{CM} is BER of the channel modulation. If the error is within an index number, then an index i is erroneously decoded as index $i-1$ or $i+1$. After decoding, two bits may therefore be wrong within a segment: the bit with index i and a bit with index $i-1$ or $i+1$. Given that there is a pulse in error, the probabilities of these four cases to occur are 0.5, $1/(n+3)$, $1/2(n+3)$, and $n/2(n+3)$, respectively, where n is the total number of ON bits in all the four encoded segments. With increasing packet size, the error probability decreases when the $CNOI$ and $CFlags$ are corrupted, increases when the indices are corrupted, but remains constant when α is corrupted. Considering BPSK modulation, we have run simulations for the maximum and minimum BER of ECS as shown in Fig. 2.16b. Regardless of the incoming data, the ECS BER will always fall between these two extrema.

2.5.5 Pulse Width and Shape

The pulse *width* is not a primary ECS parameter because the reception technique exploits the detection of *edges* in contrast with standard serial bit transfer, where the pulse width (inverse of the baud rate) is a primary parameter. ECS is therefore capable of working with a non-ideal pulse waveform, thus enabling great flexibility in pulse shaping. All pulse shapes are allowed as long as they satisfy switching and peak detection constraints and do not overlap with each other. A few examples are shown in Fig. 2.17.

2.5.6 Reliability

The reliability of ECS is an indication of the likelihood of successful transmission. ECS reliability is measured in terms of the *number of vulnerable locations (NVLs)*. An NVL is a sub-segment within the ECS packet which, if corrupted, can cause a

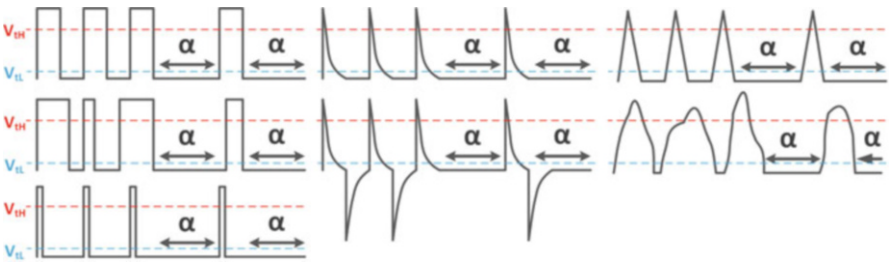

Fig. 2.17 Examples of valid signals for ECS transmission. V_{tH}: high signal level threshold. V_{tL}: low signal level threshold

packet failure. The inter-symbol separators α and the $NOIs$ are examples of $NVLs$ in ECS packet. Indeed, the longer α is, the higher the corruption likelihood, and therefore, the more likely packet failure is. Similarly, a corrupted NOI results in packet failure, and the failure likelihood increases with the number of transmitted $NOIs$. The packet failure due to these $NVLs$ has already been discussed in Sect. 2.5.3. In the ECS family of protocols, ECS1 has $NVLs$ in the range of 10–18. On the other hand, the $NVLs$ of ECS2 are reduced to a fixed number of 6, and as a result, ECS2 improves transmission reliability by about 66.7% with respect to ECS1. As for ECS, the compactness of its transmission packet restricts the $NVLs$ to the range of 5 to 12, which results in a reliability improvement of up to 72.2% with respect to ECS1 and makes it competitive with ECS2.

2.5.7 Robustness

As highlighted in Sect. 2.5.5, ECS is indifferent to pulse shape and width because only edges are used for decoding at reception. This important ECS property results in a remarkable tolerance toward clock discrepancies, jitters, and skews. A detailed analysis of these clocking and timing aspects are presented in Chap. 3. Therein, a threshold on clock rate difference between transceivers is derived, below which the protocol operates with zero decoding error over an ideal channel. It must be noted that this threshold does not define a boundary beyond which ECS needs a synchronization mechanism. It is rather an indicator when ECS needs to change its protocol parameters to enable communication with network nodes that are operating at significantly different clock rates. Chapter 6 highlights the methodology to establish a successful communication link automatically when the abovementioned threshold on clock difference is crossed. In the ideal scenario where there are no clock skews or jitters, the threshold is determined by α and α_{th}. For the minimum recommended settings of $\alpha = 4$ and $\alpha_{th} = 2$, the upper bound on clock rate difference for a 25 MHz transceiver is 10.3 MHz. Beyond this difference, it is impossible for the receiver counter on the slow transceiver to reach the threshold value of 3 or higher, as discussed in Sect. 2.5 and illustrated in Fig. 2.6. Clock jitters and skews result in the decrease of this ideal upper bound on clock rate difference. The possible ranges for clock skews and jitters are presented in Chap. 3, where a trade-off between reliability, defined as robustness with respect to clock rate variations, and data rate is quantified and used for protocol parameter selection. Using $\alpha = 4$, an ECS clock equal to the system clock of 25 MHz, and worst-case clock skew and jitter, the upper bound on clock rate difference between ECS transceivers is 4.6 MHz ($\sim$20%). In practice, this means that if one ECS end is operating at 25 MHz, the other ECS end can operate in the range [20.4 MHz, 29.6 MHz] without impacting the reliability of the transmission. One important area of future investigation is the impact of non-ideal transmission channels on ECS robustness.

2.5.8 Overall Latency

The impact of ECS on the latency of the communication system very much depends on the ECS hardware implementation. The minimum latency is one clock cycle, and to achieve it, the four data segments can be encoded together before starting transmission. The drawback of this approach is that the encoder hardware and the memory used to store encoding information would be quadrupled. The alternative that we have adopted is a pipelined datapath where only two segments are encoded before starting transmission. Indeed, at least two segments need to be encoded to generate $CNOIs$ and initiate the transmission process, as explained in Sect. 2.1.3. This pipelined architecture requires only twice the encoding and memory hardware. The same hardware is reused to encode the two segments, while the indices of the previous two segments are being transmitted. The encoder hardware is combinational, which results in a latency of one clock cycle only. On the other hand, sequential implementation of the encoding process would introduce a latency of more than one clock cycle.

2.5.9 Networking

ECS is architecturally flexible in that it supports a wide variety of networking options. It can be configured in various network topologies, including Master–Slave, Star, Ring, Tree, and Peer-to-Peer. In a single-channel implementation (Chap. 9), several MSP430 cores have been used to establish a Master–Slave network of low-end devices, as shown in Fig. 2.18a. The same setup can be used to establish a ring network, as shown in Fig. 2.18b. In the latter, where communication is restricted to nearest-neighbor devices, device IDs have been used to enforce the ring topology.

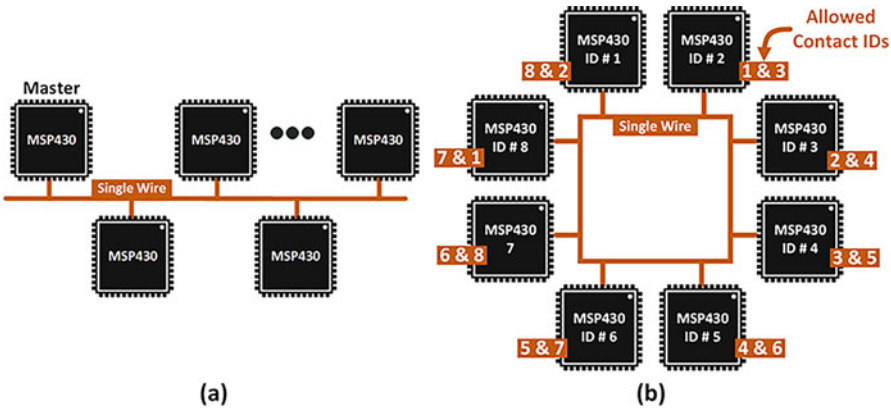

Fig. 2.18 Network topology [50]: (**a**) Master–Slave, (**b**) configuring ring

For example, MSP430 with ID number 2 can only communicate with the devices having ID numbers 1 and 3. Device IDs can similarly be used to implement a network of ECS devices without any change in the underlying hardware.

2.6 Conclusions

In conclusion, ECS is a novel, single-channel, communication protocol that simultaneously meets the requirements of low power, high data rate, reliability, and secure transmission for device-to-device communication between constrained edge nodes. ECS reduces silicon area and power consumption significantly by eliminating the need of power- and area-hungry circuits for clock and data recovery. This is because ECS packet reception and decoding are based on counting the rising edges of the transmitted pulses which makes the pulse width inconsequential. ECS is robust with respect to skews, jitters, and clock variations and combines the best features of both ECS1 and ECS2 with respect to data rate, reliability, packet security, and power efficiency. In summary, ECS is the better choice for constrained devices in a variety of use cases, including small footprint transceivers, wireless sensor nodes, implantable devices, and body-area networks. The protocol can be applied to other communication media such as photonics, infrared, and visible light.