9.7 CHECKSUM

A checksum is a summation of a set of data and can be an arbitrary width, usually 8, 16, or 32 bits. Once a set of data has been summed, the checksum is sent along with the frame so that the sum can be verified at the receiver. The receiver can calculate its own checksum value by summing the relevant data and then compare its result against the frame's checksum. Alternatively, many checksum schemes enable the checksum value itself to be summed along with the data set, with a final result of zero indicating verification and nonzero indicating an error.

Perhaps the most common checksum scheme uses one's complement binary arithmetic to calculate the sum of a data set. One's complement addition involves calculating a normal two's complement sum of two values and then adding the carry bit back into the result. This constrains the running sum to the desired bit width, a necessary feature when summing hundreds or thousands of bytes where an unconstrained sum can be quite large. It is guaranteed that, when the carry bit is added back into the original result, a second carry will not be generated. For example, adding 0xFF and 0xFF yields 0x1FE. When the carry bit is added back into the eight-bit sum, 0xFF is the final one's complement result.

An interesting consequence of one's complement math is that the eight-bit values 0x00 and 0xFF (or 0xFFFF for a 16-bit value) are numerically equivalent. Consider what would happen if 1 is added to either value. In the first case, 0x00 + 0x01 = 0x01, is the obvious result. In the second case, 0xFF + 0x01 = 0x100 = 0x01, where the carry bit is added back into the eight-bit sum to yield the same final result, 0x01. A brief example of calculating a 16-bit checksum is shown in Table 9.6 to aid in understanding the one's complement checksum.

TABLE 9.6 Sixteen-Bit One's Complement Checksum

Data Value	Sum	Carry	Running Checksum
Initialize checksum to zero			0x0000
0x1020	0x1020	0	0x1020
0xFFF0	0x1010	1	0x1011
0xAD00	0xBD11	0	0xBD11
0x6098	0x1DA9	1	0x1DAA
0x701E	0x8DC8	0	0x8DC8

The logic to perform a one's complement checksum calculation can take the form of a normal two's complement adder whereby the carry bit is fed back in the next clock cycle to adjust the sum. As shown in Fig. 9.9, this forms a pipelined checksum calculator where the latency is two cycles. To begin the calculation, the accumulator and carry bit are reset to 0. Each time a word is to be summed, the multiplexer is switched from 0 to select the word. The adder has no qualifying logic and adds its two inputs, whose sum is repeatedly loaded into the accumulator and carry bits on each rising clock edge. After the last word has been summed, the control logic should wait an extra cycle, during which the multiplexer is selected to 0 to allow the most recent carry bit to be incorporated into the sum. It is guaranteed that a nonzero carry bit will not propagate into another nonzero carry bit after summing the accumulator with 0. This configuration works well in many situations, because two's complement adders are supported by many available logic implementation technologies. In high-

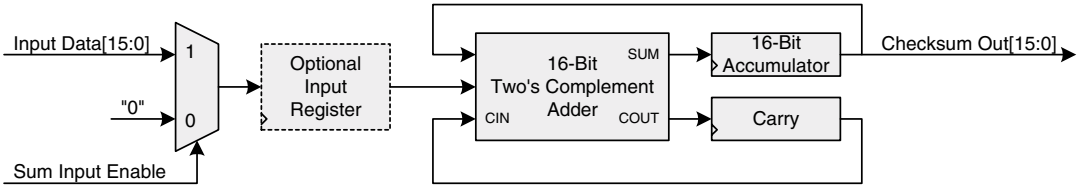

FIGURE 9.9 One's complement checksum calculator.

speed designs, logic paths with adders in them can prove difficult to meet timing. As shown in Fig. 9.9, an optional input register can be added between the multiplexer and the adder to completely isolate the adder from the control logic. This modification improves timing at the expense of an added cycle of latency to the checksum calculation.

A convenient advantage to the one's complement checksum is that, if the final result is inverted, or complemented, it can be summed along with the original data set with a final result of 0. The complement of 0x8DC8 is 0x7237, which, when added to the sum, 0x8DC8, yields 0xFFFF, the equivalent of 0x0000. Therefore, rather than placing the checksum itself into a frame, the transmitter can place the complemented value instead. The receiver's verification logic is thereby simplified by not requiring a full 16-bit comparator between the calculated checksum and the frame's checksum. It can sum all the relevant data values along with the complemented checksum and check for a result of 0x0000 or 0xFFFF.

Checksums can catch many types of errors, because most errors will cause the receiver's calculated checksum to differ from that contained in the frame. There are, however, consecutive sequences of errors, or bursts, that a checksum cannot detect. As the checksum is calculated, the first bit error will corrupt the in-progress sum. Subsequent errors can mask the previous error and return the sum to its proper value. The probability of this occurring is low, but it is not zero. Error detection is a study in probability wherein one can never attain a zero probability of undetected errors, but the probability can be brought arbitrarily close to zero as overhead is added in the form of coding and error detection fields.

9.8 CYCLIC REDUNDANCY CHECK

The CRC is a more complex calculation and one that provides a higher probability of detecting errors, especially multibit burst errors. A CRC is calculated using a linear feedback shift register, following the same basic concept behind scrambling data with a polynomial according to Galois field theory. CRCs exhibit superior error detection characteristics including the ability to detect all single-bit and double-bit errors, all odd numbers of errors, all burst errors less than or equal to the degree of the polynomial used, and most burst errors greater than the degree of the polynomial used.* The quality of CRC error detection depends on choosing the right polynomial, called a *generator polynomial*. The theory behind mathematically proving CRC validity and choosing generator polynomials is a complex set of topics about which much has been written. Different standard applications that employ CRCs have a specific polynomial associated with them.

A common CRC algorithm is the 8-bit polynomial specified by the *International Telecommunication Union* (ITU) in recommendation I.432, and it is used to protect ATM cell headers. This CRC,

* *Parallel Cyclic Redundancy Check (CRC) for HOTLink™,* Cypress Semiconductor, 1999.