142.1.1 Conventions

142.1.1.1 State diagrams

The body of this standard comprises state diagrams, including the associated definitions of variables, constants, and functions. The notation used in the state diagrams follows the conventions in 21.5. In case of any discrepancies between a state diagram and descriptive text, the state diagram prevails.

142.1.1.2 Timers

Some state diagrams may utilize timers. Timers follow the conventions of 14.2.3.2 augmented as follows:

a) [start x_timer, y] sets expiration of *y* to timer *x_timer*.
b) Upon expiration of timer *x_timer*, a Boolean variable *x_timer_done* gets asserted automatically. Restarting the timer *x_timer* deasserts the value of *x_timer_done*.
c) [stop x_timer] aborts the timer operation for *x_timer* deasserting *x_timer_done* indefinitely.

142.1.1.3 Operations on variables

The state diagram operators are shown in Table 142-X.

Table 142-x—State diagram operators

Operator	Meaning
AND	Logical or bitwise AND. If both operands are defined as Boolean values, the operation is logical AND. Otherwise, the operation is considered the bitwise AND (each bit of the first operand is logically AND-ed with the corresponding bit of the second operand).
OR	Logical or bitwise OR. If both operands are defined as Boolean values, the operation is logical OR. Otherwise, the operation is considered the bitwise OR (each bit of the first operand is logically OR-ed with the corresponding bit of the second operand).
XOR	Logical or bitwise exclusive OR. If both operands are defined as Boolean values, the operation is logical XOR. Otherwise, the operation is considered the bitwise XOR (each bit of the first operand is logically XOR-ed with the corresponding bit of the second operand).
!	Boolean NOT
⇐	Assignment operator
<	Less than (see 142.1.14)
>	More than (see 142.1.14)
≤	Less than or equal to (see 142.1.14)
≥	More than or equal to (see 142.1.14)
=	Equals (a test of equality)
!=	Not equals
++	Unary opperator placed after a variable; increents the variable by 1
--	Unary opperator placed after a variable; decreents the variable by 1
+=	Increments left operand value by the value of the operand on the right (x += y is equivalent to x ⇐ x+y)
-=	Decrements left operand value by the value of the operand on the right (x -= y is equivalent to x ⇐ x-y)
( )	Indicates precedence or a set of function arguments
|	Concatenation operation that combines several subfields or parameters into a single aggregated field or parameter

Variables that allow access to individual bits are called *vectors*. The vector notations use 0 to mark the first received bit. Individual bits are accessed using the following notation:

a) data_vector<k> accesses k^{th} bit of the vector.
b) data_vector<m:n> accesses bits *n* through *m* inclusively. The n^{th} bit is received earlier than the m^{th} bit.

Refer to 3.1 for the conventions on bit ordering.

142.1.1.4 Comparisons of cyclic variables

A function $a < b$ is used to compare two values. Returned value is true when *b* is larger than *a* allowing for wrap-around of *a* and *b*. The comparison is made by subtracting *b* from *a* and testing the MSB. When MSB($a–b$) = 1 the value *true* is returned, else *false* is returned. In addition, the following functions are defined in terms of $a < b$:

a) $a > b$ is equivalent to !($a < b$ *or* $a = b$)
b) $a \geq b$ is equivalent to !($a < b$)
c) $a \leq b$ is equivalent to !($a > b$)

142.1.1.5 FIFO access operations

State diagrams used in this standard make extensive use of *first-in, first-out* (FIFO) buffers. These buffers support a common set of operations, as defined below:

a) *Buf.Append(e)* adds the element *e* to the input of FIFO buffer *Buf*.
b) *Buf.Clear()* removes all elements from the FIFO buffer *Buf*.
c) *Buf.Fill(e)* writes element/value *e* into each position of FIFO buffer *Buf*.
d) *Buf.GetHead()* returns the oldest (head) element in the FIFO buffer Buf, and removes that element from the FIFO, decreasing its length by one.
e) *Buf.IsEmpty()* returns true if the FIFO buffer is empty (has no elements), otherwise the function returns false.
f) *Buf.IsFull()* returns true if the FIFO buffer *Buf* is full (i.e., *Buf* has no unoccupied positions), otherwise the function returns false.
g) *Buf.PeekHead()* returns the oldest (head) element in the FIFO buffer *Buf* without removing that element from the FIFO.

All of the FIFO access operations are assumed to be non-blocking and to take zero time to complete the execution.