State diagram conventions and logical operators

Glen Kramer, Broadcom

Initial SD conventions (SC 1.2.1)

- Initial state diagram conventions were defined in subclause 1.2.1
- These conventions are mostly obsolete now, though some clauses still reference 1.2.1

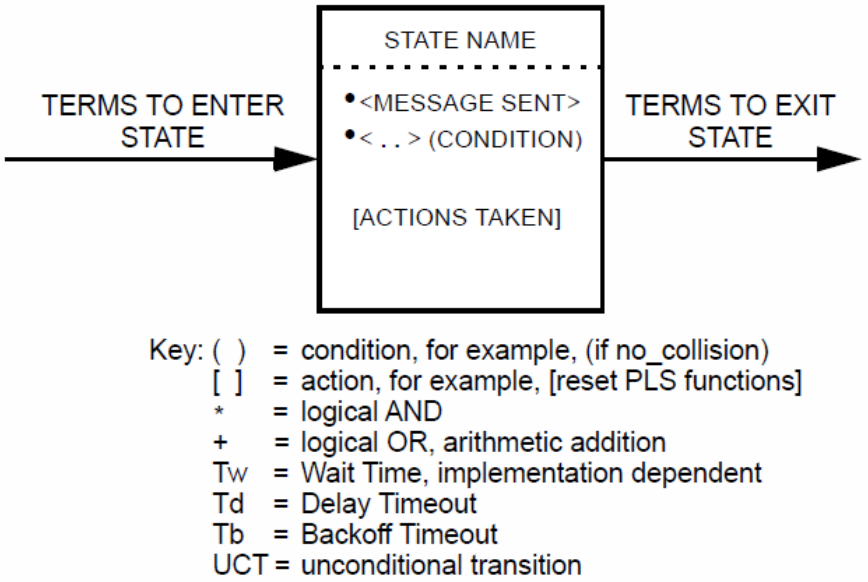
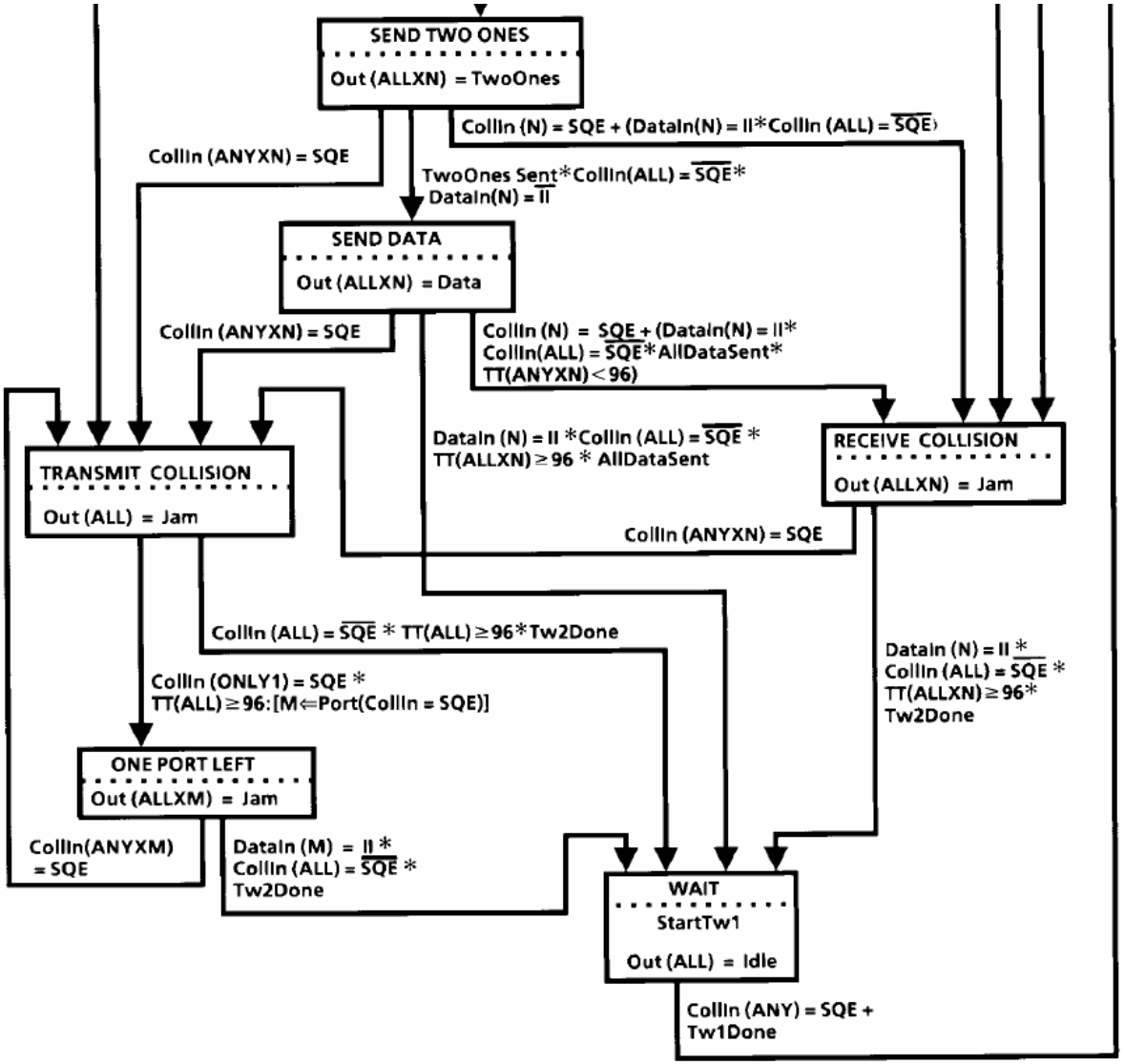


Figure 1–2—State diagram notation example

a) The character ":" (colon) is a delimiter used to denote that a term assignment statement follows.
b) The character "⇐" (left arrow) denotes assignment of the value following the arrow to the term preceding the arrow.

Example of SD according to 1.2.1

- Both the notation and look and feel are significantly different from what we have in more modern clauses.

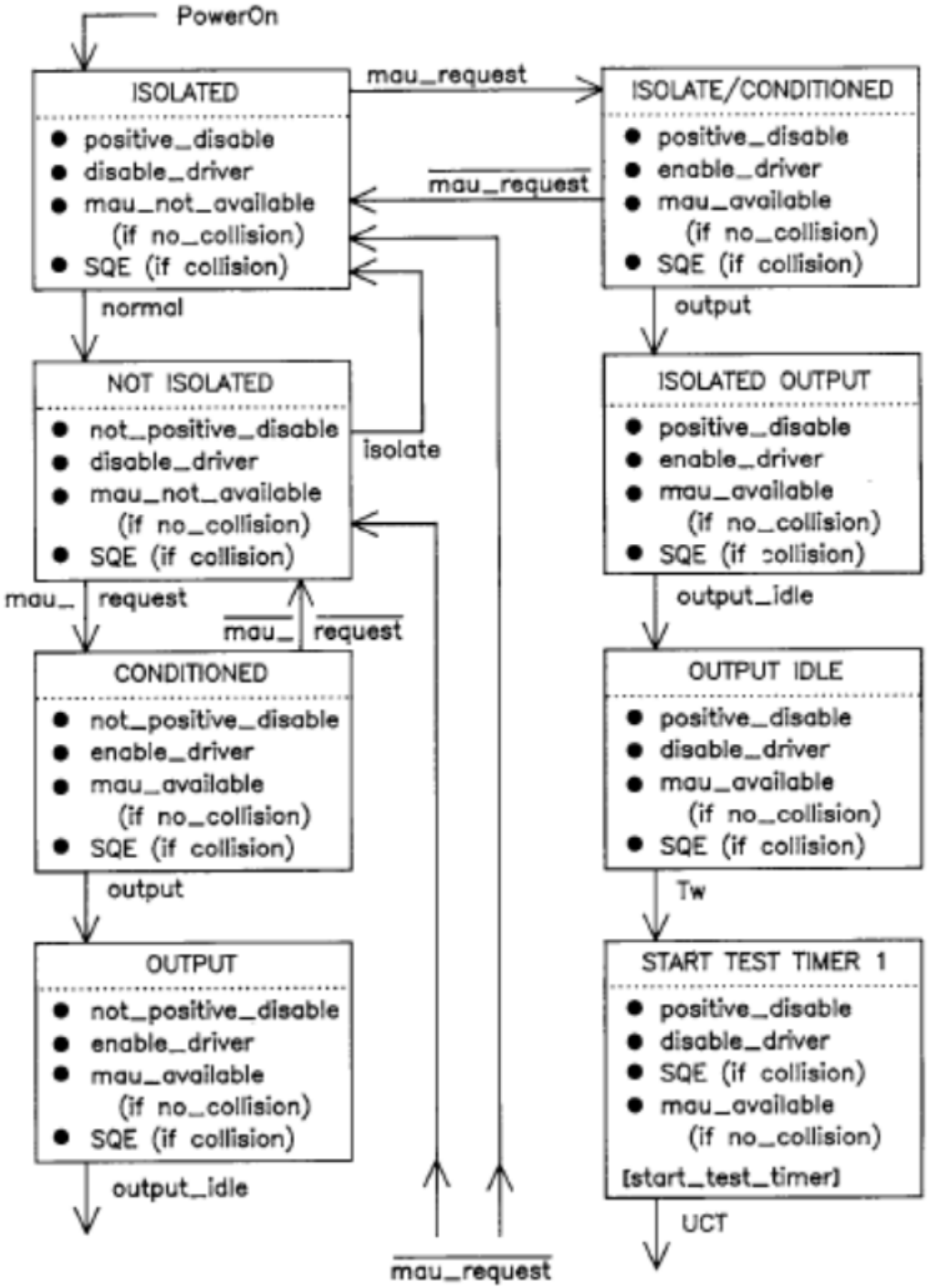


NOTE: Out (X) = Idle in all instances unless specified otherwise.

Figure 9–2—Repeater unit state diagram

Another example

- Bulleted lists in states



Updated conventions (SC 21.5)

- Clause 21.5 has redefined some of the state diagram conventions
- Interestingly, conventions in 21.5 do not list any operators for arithmetic addition, subtraction, multiplication, and division.
  - "+" and "*" are only shown as logical OR and AND.

Table 21–1—State diagram operators

Character	Meaning
*	Boolean AND
+	Boolean OR
^	Boolean XOR
!	Boolean NOT
<	Less than
≤	Less than or equal to
=	Equals (a test of equality)
≠	Not equals
≥	Greater than or equal to
>	Greater than
( )	Indicates precedence
⇐	Assignment operator
∈	Indicates membership
∉	Indicates nonmembership
|	Catenate
ELSE	No other state condition is satisfied

No unified conventions in 802.3

- Few old clauses reference conventions in 1.2.1.
- Most clauses reference conventions in 21.5.
- Some clauses reference multiple locations.

34.2 State diagrams

State diagrams take precedence over text.

The conventions of 1.2 are adopted, along with the extensions listed in 21.5.

35.4.2.1 Conventions

The notation used in the state diagram follows the conventions of 34.2.

50.1.7 Notational conventions

The state diagrams within the body of this clause follow the notations and conventions of 21.5. State diagram timers follow the conventions of 14.2.3.2.

- Very many clauses add extensions or exceptions to the existing conventions. This is true for all EPON clauses that contain state diagrams (see 64.1.5, 65.2.2.2, 76.1.1, 77.1.5)

State diagram notation issues

- Many clauses directly or indirectly reference SD conventions in 21.5.
  - "+" is defined to mean "logical OR". Most state diagrams use it for both logical OR and arithmetic addition.
  - "*" is defined to mean "logical AND". Most state diagrams use it for both logical AND and arithmetic multiplication.
- Such dual use of "+" and "*" leads to confusion.
- Consider these two examples from 802.3ca
  - Will a reader understand that one transition assumes additions, while the other assumes logical OR operations?

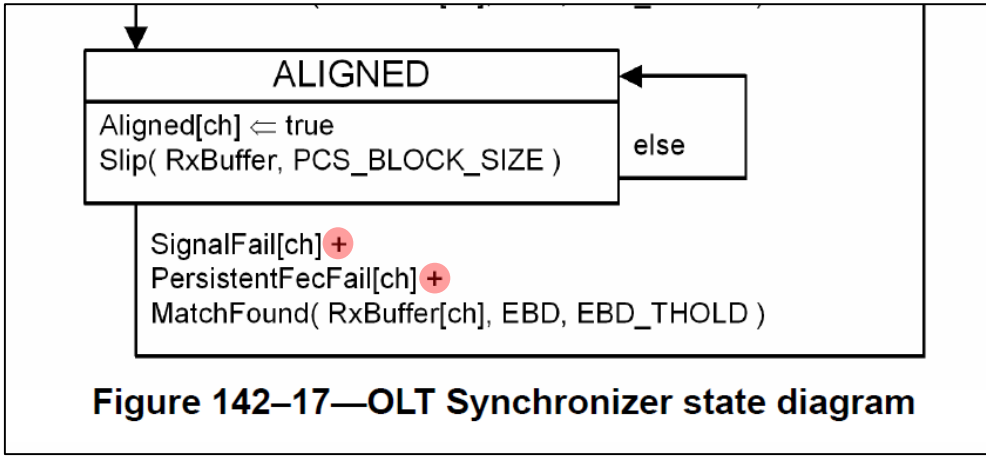


Figure 142–17—OLT Synchronizer state diagram

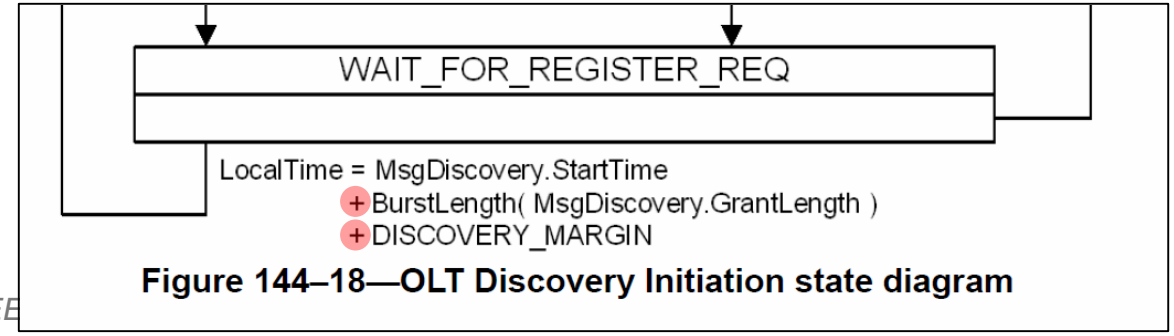


Figure 144–18—OLT Discovery Initiation state diagram

Other projects fixed this

- For example, 1904.1 switched to using **OR**, **AND**, and **XOR**.
- Comment #412 updates SD conventions for all 802.3ca clauses
  - Adds definitions for FIFO operations
  - Clarifies vector notations
  - Defines "++", "—", "+=", and "-=" operators.
- Is it time we also fix "+" and "-" ambiguity in 802.3ca?

Character	Meaning
AND	Boolean AND
OR	Boolean OR
XOR	Boolean XOR
!	Boolean NOT
<	Less than
>	More than
≤	Less than or equal to
≥	More than or equal to
==	Equals (a test of equality)
!=	Not equals
()	Indicates precedence

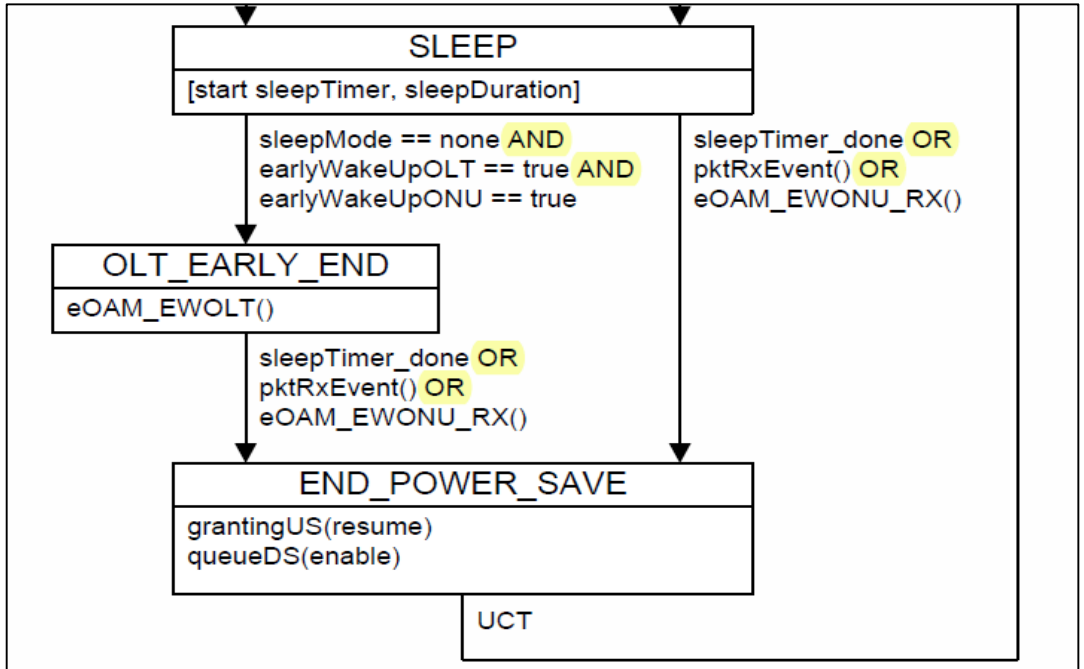


Figure 10-6—Power-saving mechanism state diagram in the OLT

No consistent notation in 802.3

- **|** is used throughout the standard (including .3ca) to represent concatenation. Also used to represent absolute values, i.e., |A-B|
- C27 uses **&** for concatenation
- C55 and C113 use **&** and **XOR** for bitwise AND and XOR
- C144 uses **&** for bitwise AND
- C40 and C91 use **^** for XOR
- C61A, C101 use standard C/C++ operators
  - **|** - bitwise OR
  - **&** - bitwise AND
  - **^** - bitwise XOR
  - **||** - logical OR
  - **&&** - logical AND
- Many MATLAB code examples use **^** for “taken to the power of”

Proposal

- Use AND, OR, XOR
- Logical vs. Bitwise is deduced from the type of operands

Operator	Meaning
AND	Logical or bitwise AND. If one or 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 one or 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 one or 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).

Thank You