Technical Descriptions for Cut-Through Forwarding in Bridges

DCN 1-22-0042-00-ICne

Author: Johannes Specht

September 8, 2022

Contents

List of Figures

1 Introduction

1.1 Purpose

This document is an individual contribution by the author, provided for technical discussion in pre-PAR activities of IEEE 802 (i.e., Nendica). The contents of this document are technical descriptions for the operations of Cut-Through Forwarding (CTF) in bridges. The intent is to provide more technical clarity, and thereby also address the desire expressed by some individuals during the IEEE 802 Plenary Meeting in July 2022 to a certain extent.

1.2 Relationship to Standards

This document **IS NOT** an IEEE Standard or an IEEE Standards draft. This allows readers to focus on the technical contents in this document, rather than additional aspects that are important during standards development. For example:

1. The structure of this document does not comply with the structural requirements for such standards. For example, it does not contain mandatory clauses for IEEE Standards [2].
2. Usage of normative keywords has no implied semantics beyond explicit description. For example, usage of the words *shall*, *should* or *may* **DOES NOT** imply requirements or recommendations for conformance of an implementation.
3. This document contains references, but without distinguishing between normative and informative references.
4. This document does not contain suggestions for assigning particular contents to *vehicles* (e.g., IEEE 802 Working Groups, potential amendment projects for existing standards, or potential new standard projects). As a consequence, the clause structure of this document is intended for readability, rather than fitting into the clause structure of a particular Standard (i.e., which would matter for potential amendment projects).

1.3 Status of this Document

This document is incomplete and contains technical and editorial errors and omissions. Readers discovering such issues may sent enhancement proposals to the author via email (johannes.specht.standards@gmail.com).

2 Generalized Receive Operations

2.1 Overview

The generalized receive operations are described by a stack of processes that interact via global variables (see 2.4) and service primitive invocations (see 2.2). Figure 2.1 provides an overview of these processes and their interaction[1]. The processes can be

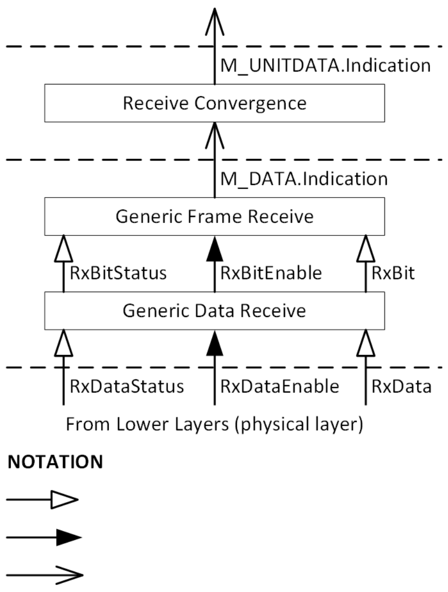


Figure 2.1: Overview of the Generic Serial Receive Function.

summarized as follows:

1. A Generic Data Receive Process (see 2.5) that translates a lower layer-dependent[2] serial data stream into a uniform bit stream.
2. A Generic Frame Receive Process (see 2.6) that generates M_DATA.indication invocations from the uniform bit stream.

[1]This interaction model is inspired by clause 6 and 8.6.9 of IEEE Std 802.1Q[5].
[2]Such a lower layer may be an entity on the physical layer (PHY), but the generalized receive operations are not limited to this.

3. A Receive Convergence Process (2.7) that translates M_DATA.indication invocations into M_UNITDATA.indication primitives.

The generalized receive operations are inspired by the concepts of a generic serial convergence functions as described in slides by Roger Marks [1, slide 15].

The generalized receice operations follow a different modelling approach with more formalized description of these functions and incorporate some of the following concepts, as suggested by the Author of this document during the Nendica meetings on and after August 18, 2022. The differences can be summarized as follows:

- Alignment with the state machine diagram conventions in Annex E of IEEE Std 802.1Q[5].
- Support for serial data streams from lower layers with arbitrary data word length[3].
- Explicit modelling of start and end of atomic ISS service primitive invocations (see also 7.2 and 11.1 of IEEE Std 802.1AC[4]).

By keeping ISS service primitive invocations atomic, the approach in this document is intended to provide a higher level of compatibility with existing IEEE 802.1 Stds, similar to the modelling approach via frame look-ahead of service primitive invocations/prescient functions[3, slides 7ff.].

2.2 Service Primitives

2.2.1 M_DATA.indication

The M_DATA.indication service primitive passes the contents of a frame from the Generic Frame Receive process to the Receive Convergence process. This indication has the following parameter signature:
M_UNITDATA.indication (DA, SA, SDU, FCS)

The parameters of the M_DATA.indication are defined as follows:

DA An array of zero to LEN_ADDR(2.3.2) bits, containing the destination address of a frame.

SA An array of zero to LEN_ADDR(2.3.2) bits, containing the source address of a frame.

SDU An array of zero or more bits, containing a service data unit of a frame. The number of bits after complete reception of a frame is an integer multiple of eight.

FCS An array of zero to LEN_FCS(2.3.3) bits, containing the frame check sequence of a frame.

[3]This generalization is intended to allow a wide range of lower layers. In addition, the support for word sizes (e.g., 8 bits, 32 bits or 64 bits) may be close to realities found in hardware implementation. It is subject to discussion whether this and other generalizations over [1] introduced by the author are considered to be helpful.

2.2.2 M_UNITDATA.indication

As specified in 11.1 of IEEE Std 802.1AC[4].

2.2.3 Atomic Invocation Model

All invocations of service primitives in this document are atomic. That is, each invocation in non-dividable (see also 7.2 of IEEE Std 802.1AC[4]). Service primitive invocations can be modeled more explicitly to allow for accurate description of operations within a Bridge. This explicit model comprises the following:

1. A service primitive provides two attributes, *'start* and *'end*. These attributes are used in subsequent descriptions to indicate the start and the end of the indication, respectively.
2. The parameters of a service primitive are explicitly modeled as bit arrays.
3. The values of parameters during invocations of a service primitive are passed according to a call-by-reference scheme.

2.3 Global Constants

2.3.1 PREAMBLE

A lower layer-dependent array of zero[4] or more bits, containing the expected preamble of each frame.

2.3.2 LEN_ADDR

An integer denoting the length of the DA and SA parameters of M_DATA.indication parameters, in bits. For EUI-48 addresses, LEN_ADDR is 48.

2.3.3 LEN_FCS

An integer denoting the length of frame check sequence and the length FCS parameter of M_DATA.indication parameter, respectively, in bits. For example,

$$\text{LEN\_FCS} = 32$$

indicates a four octet frame check sequence.

[4]Including length zero permits to support lower layers that do not expose a preamble to the Generic Data Receive process.

2.3.4 LEN_MIN

A lower layer-dependent integer, denoting the minimum length of a frame, in bits. Invocation of the The M_DATA.indication service primitive starts once the Generic Frame Receive process received the first LEN_MIN bits of a frame. Any value for LEN_MIN greater than 0 is valid.

2.3.5 LEN_MAX

A lower layer-dependent integer, denoting the maximum length of a frame, in bits. Invocation of the The M_DATA.indication service primitive ends at latest once the Generic Frame Receive process received at most LEN_MAX bits of a frame. Any value for LEN_MAX greater than LEN_MIN is valid.

2.3.6 LEN_DATA

A lower layer-dependent integer, denoting the width of the RxData variable, in bits.

2.4 Global Variables

2.4.1 RxBitEnable

A Boolean variable, set by the Generic Data Receive process and reset by the Generic Frame Receive process, which indicates an update of the RxBit variable, RxBitStatus variable, or both.

2.4.2 RxBit

A bit variable used to pass a single bit value to the Generic Frame Receive Process.

2.4.3 RxBitStatus

An enumeration variable used to pass the receive status from the Generic Data Receive process to the Generic Frame Receive process. The valid enumeration literals are as follows:

RECEIVING Indicates that the Generic Data Receive process received data from lower layers in a serial stream without knowledge of the remaining length of the overall data stream.

TRAILER Indicates that the Generic Data Receive process received data from lower layers in a serial stream with the knowledge that LEN_FCS or less bits follow.

2.4.4 RxDataEnable

A Boolean variable, set by a lower layer and reset by the Generic Data Receive process, which indicates an update of the RxData variable, RxDataStatus variable, or both.

2.4.5 RxData

An lower layer-dependent array or of LEN_DATA bits, used to pass a single data word to the Generic Data Receive Process.

2.4.6 RxDataStatus

An enumeration variable used to pass the receive status from lower layers to the Generic Data Receive process. The valid enumeration literals are as follows:

RECEIVING Indicates that data stream reception from lower layers is active.

IDLE Indicates that data stream reception from lower layers is not active.

2.5 Generic Data Receive

The Generic Data Receive process translates a lower layer-dependent[5] serial data stream into a uniform bit stream. In addition, it realizes the following functions:

- Hide frames with less than LEN_MIN (2.3.4) bits from the Generic Frame Receive Process (2.6).
- Determine the position in the serial data stream of a frame at which the frame check sequence begins (delay line modelling).
- Truncate excess bits to satisfy the frame length requirements implied by the parameter definition of the M_DATA.indication primitive (2.2.1).

2.6 Generic Frame Receive

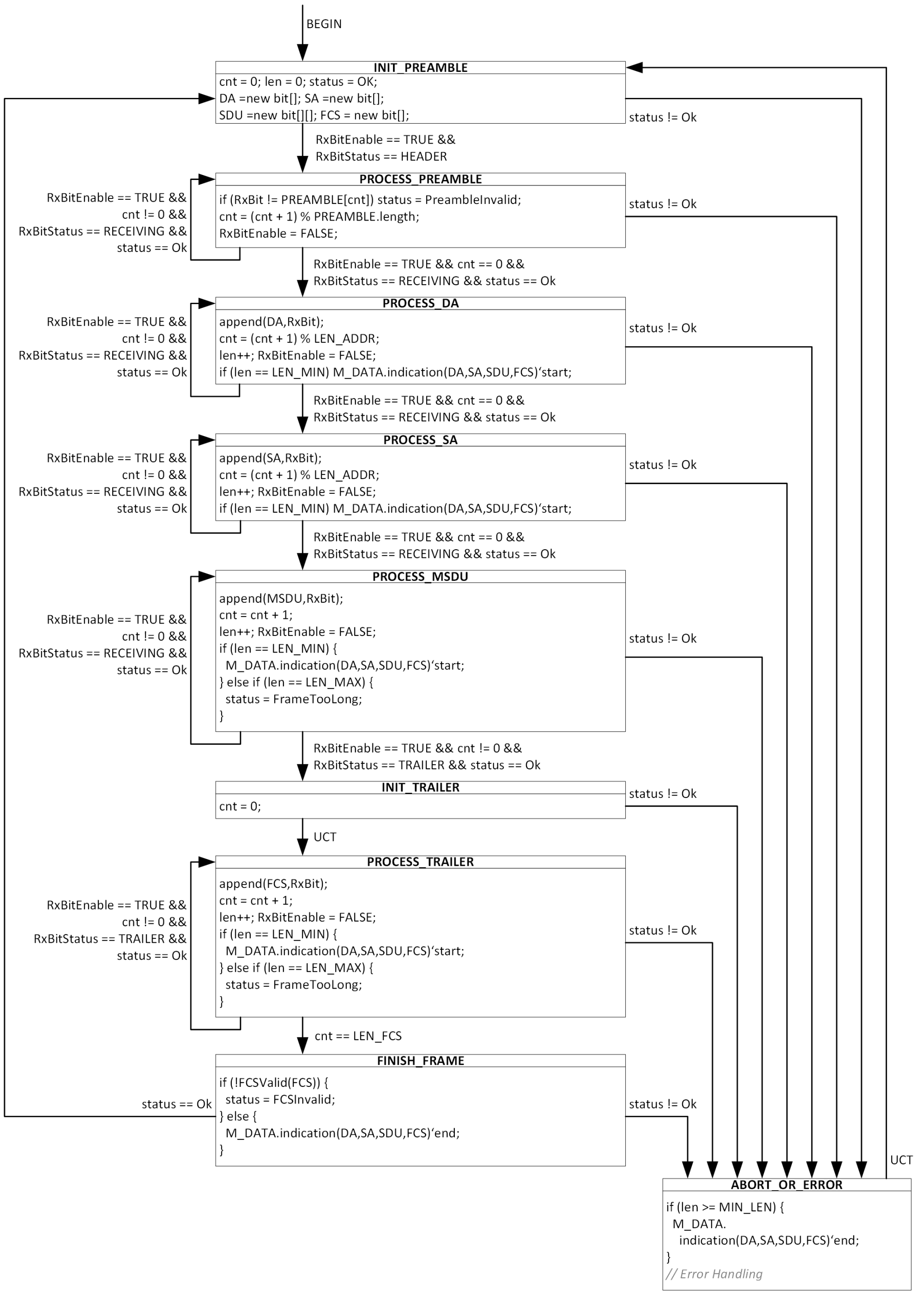
2.6.1 Description

The Generic Frame Receive process transforms a serial bit streams of frames from the Generic Data Receive process into invocations of the M_DATA.indication primitive.

2.6.2 State Machine Diagram

The operation of the Generic Frame Receive process is specified by the state machine diagram in Figure 2.2 , using the variables and functions defined in subsequent subclauses.

[5]Such a lower layer may be an entity on the physical layer (PHY), but the generalized receive operations are not limited to this.


BEGIN
RxBitEnable == TRUE && RxBitStatus == HEADER
RxBitEnable == TRUE && cnt != 0 && RxBitStatus == RECEIVING && status == Ok
RxBitEnable == TRUE && cnt == 0 && RxBitStatus == RECEIVING && status == Ok
RxBitEnable == TRUE && cnt != 0 && RxBitStatus == TRAILER && status == Ok
UCT
RxBitEnable == TRUE && cnt != 0 && RxBitStatus == TRAILER && status == Ok
cnt == LEN_FCS
status == Ok
status != Ok
UCT

```
INIT_PREAMBLE
cnt = 0; len = 0; status = OK;
DA =new bit[]; SA =new bit[];
SDU =new bit[][]; FCS = new bit[];
```

```
PROCESS_PREAMBLE
if (RxBit != PREAMBLE[cnt]) status = PreambleInvalid;
cnt = (cnt + 1) % PREAMBLE.length;
RxBitEnable = FALSE;
```

```
PROCESS_DA
append(DA,RxBit);
cnt = (cnt + 1) % LEN_ADDR;
len++; RxBitEnable = FALSE;
if (len == LEN_MIN) M_DATA.indication(DA,SA,SDU,FCS)'start;
```

```
PROCESS_SA
append(SA,RxBit);
cnt = (cnt + 1) % LEN_ADDR;
len++; RxBitEnable = FALSE;
if (len == LEN_MIN) M_DATA.indication(DA,SA,SDU,FCS)'start;
```

```
PROCESS_MSDU
append(MSDU,RxBit);
cnt = cnt + 1;
len++; RxBitEnable = FALSE;
if (len == LEN_MIN) {
  M_DATA.indication(DA,SA,SDU,FCS)'start;
} else if (len == LEN_MAX) {
  status = FrameTooLong;
}
```

```
INIT_TRAILER
cnt = 0;
```

```
PROCESS_TRAILER
append(FCS,RxBit);
cnt = cnt + 1;
len++; RxBitEnable = FALSE;
if (len == LEN_MIN) {
  M_DATA.indication(DA,SA,SDU,FCS)'start;
} else if (len == LEN_MAX) {
  status = FrameTooLong;
}
```

```
FINISH_FRAME
if (!FCSValid(FCS)) {
  status = FCSInvalid;
} else {
  M_DATA.indication(DA,SA,SDU,FCS)'end;
}
```

```
ABORT_OR_ERROR
if (len >= MIN_LEN) {
  M_DATA.
    indication(DA,SA,SDU,FCS)'end;
}
// Error Handling
```

Figure 2.2: State Machine Diagram of the Generic Frame Receive Process.

2.6.3 Variables

2.6.3.1 cnt

An integer counter variable, used to count the number of bits in the current parameter of the frame.

2.6.3.2 len

An integer variable holding the actual length of a frame under reception, in bits.

2.6.3.3 status

An enumeration variable holding the current status of the Generic Frame Receive process. The valid enumeration literals are as follows:

Ok Indicates that no error has been discovered prior or during frame reception.

FrameTooLong Indicates that a frame under reception exceeded LEN_MAX bits.

FCSInvalid Indicates inconsistency between the FCS parameter an the remaining parameters of a frame under reception.

2.6.4 Functions

2.6.4.1 append(parameter,bit)

The append function appends a given bit at the end of a particular parameter of an M_DATA.indication service primitive.

2.6.4.2 FCSValid(FCS)

The FCSValid function determines if the FCS parameter consistent with the remaining parameters of the M_DATA.indication service primitive (TRUE) or not (FALSE).

2.7 Receive Convergence

The Receive Convergence Process implements the translation of M_DATA.indication invocations to M_UNITDATA.indication invocations. The supported translations are lower layer-dependent and include, but are not limited to, those specified in clause 13 of IEEE Std 802.1AC[4].

Each M_DATA.indication invocation results in an associated M_UNITDATA.-indication invocation. During the translation, the M_UNITDATA.indication parameters are extracted from the M_DATA.indication parameters according to the rules defined for the underlying medium.

Bibliography

[1] Roger Marks (EthAirNet Associates). *Generic Serial Convergence Function (GSCF).*

[2] IEEE Standards Association. *2021 IEEE SA Standards Style Manual.*

[3] Inc.) Johannes Specht (Self; Analog Devices, Inc.; Mitsubishi Electric Corporation; Phoenix Contact GmbH & Co. KG; PROFIBUS Nutzerorganisation e.V.; Siemens AG; Texas Instruments. *CTF - Considerations on Modelling, Compatibility and Locations.*

[4] LAN/MAN Standards Committee of the IEEE Computer Society. *IEEE Std 802.1AC™-2016, IEEE Standard for Local and metropolitan area networks – Media Access Control (MAC) Service Definition.*

[5] LAN/MAN Standards Committee of the IEEE Computer Society. *IEEE Std 802.1Q™-2018, IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks.*