Meaning of terms that may be used in this document / Notice to readers

**WARNING**

Warning notices are used to emphasize that hazardous voltages, currents, temperatures, or other conditions that could cause personal injury exist or may be associated with use of a particular equipment.

In situations where inattention could cause either personal injury or damage to equipment, a Warning notice is used.

**Caution**

Caution notices are used where there is a risk of damage to equipment for example.

**Note**

Notes merely call attention to information that is especially significant to understanding and operating the equipment.

This document is based on information available at the time of its publication. While efforts have been made to be accurate, the information contained herein does not purport to cover all details or variations in hardware or software, nor to provide for every possible contingency in connection with installation, operation, or maintenance. Features may be described herein which are not present in all systems. ALSTOM assumes no obligation of notice to holders of this document with respect to changes subsequently made.

ALSTOM makes no representation or warranty, expressed, implied, or statutory with respect to, and assumes no responsibility for the accuracy, completeness, sufficiency, or usefulness of the information contained herein. ALSTOM gives no warranties of merchantability or fitness for purpose.

In this publication, no mention is made of rights with respect to trademarks or tradenames that may attach to certain words or signs. The absence of such mention, however, in no way implies there is no protection.

Partial reproduction of this document is authorized, but limited to internal use, for information only and for no commercial purpose. However, such authorization is granted only on the express condition that any partial copy of the document bears a mention of its property, including the copyright statement.

All rights reserved.
© Copyright 2001. ALSTOM (Paris, France)
## Revisions

<table>
<thead>
<tr>
<th>Index letter</th>
<th>Date</th>
<th>Nature of revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>02–1997</td>
<td>Release</td>
</tr>
<tr>
<td>c</td>
<td>01–1999</td>
<td>Addition of new functions</td>
</tr>
<tr>
<td>d</td>
<td>04–2000</td>
<td>ALSTOM branding of manual – Update</td>
</tr>
<tr>
<td>e</td>
<td>01–2001</td>
<td>Medium redundancy validation functionality</td>
</tr>
<tr>
<td>f</td>
<td>07–2001</td>
<td>Redundancy Validation Function</td>
</tr>
</tbody>
</table>
1. PURPOSE OF MANUAL AND DOCUMENTED VERSION

This manual describes the MICROFIP HANDLER software primitives used to:

- initialize the MICROFIP chip,
- manage inputs/outputs as well as communications and messages,
- process events.

2. CONTENT OF THIS MANUAL

The content of this manual is described below:

Chapter 1: Introduction: Short description of the MICROFIP HANDLER objectives and MICROFIP overview.

Chapter 2: General description of software: Description of the MICROFIP HANDLER software and its functions.

Chapter 3: Description of user interface primitives: Previous–version primitives as well as additional new primitives are described.

Chapter 4: Installation: Description of the compiling options.

Appendix A: Protocole parameter setting.

Appendix B: Example of system configuration and application.

Appendix C: Example of system configuration and application with use of the new functions.

3. RELATED PUBLICATIONS

For more information, refer to these publications:

- ALS 50280 MICROFIP User Reference Manual

4. WE WELCOME YOUR COMMENTS AND SUGGESTIONS

ALSTOM strives to produce quality technical documentation. Please take the time to fill in and return the "Reader’s Comments" page if you have any remarks or suggestions.
Your main job is:

☐ System designer  ☐ Programmer
☐ Distributor  ☐ Maintenance
☐ System integrator  ☐ Operator
☐ Installer  ☐ Other (specify below)

If you would like a personal reply, please fill in your name and address below:

COMPANY: ..........................  NAME: .................................

ADDRESS: .................................................................

...........................................  COUNTRY: ........................

Send this form directly to your ALSTOM sales representative or to this address:

ALSTOM Technology
Technical Documentation Department (TDD)
23–25 avenue Morane Saulnier
92364 Meudon la Forêt Cedex
France
Fax: +33 (0)1 46 29 10 21

All comments will be considered by qualified personnel.

REMARKS

Continue on back if necessary.
CHAPTER 1 – INTRODUCTION

1. OBJECTIVES .......................................................... 1–1

2. MICROFIP OVERVIEW .................................................. 1–2

CHAPTER 2 – GENERAL SOFTWARE DESCRIPTION

1. INTRODUCTION .......................................................... 2–1

2. FUNCTION DEFINITIONS ............................................... 2–2

CHAPTER 3 – DESCRIPTION OF USER INTERFACE PRIMITIVES

1. INTRODUCTION .......................................................... 3–1

2. LIST OF PRIMITIVES TO BE CALLED BY THE USER .......... 3–2
   2.1. mf_initialize_network() ........................................... 3–4
   2.2. mf_new_initialize_network() ...................................... 3–9
   2.3. mf_mps_var_conf() ................................................. 3–12
   2.4. mf_read_interrupt_register() .................................... 3–14
   2.5. mf_read_event() ................................................... 3–15
   2.6. mf_new_read_event() .............................................. 3–16
   2.7. mf_var_read_loc() ................................................. 3–17
   2.8. mf_var_write_loc() ............................................... 3–18
   2.9. mf_read_message() ................................................ 3–19
   2.10. mf_send_message() ............................................... 3–20
   2.11. mf_purge_send_message() ....................................... 3–21
   2.12. mf_send_message_after_purge() ............................... 3–22
   2.13. mf_reset_line_driver() .......................................... 3–23
   2.14. mf_change_time_out_value() .................................... 3–24
   2.15. mf_disable_it() .................................................. 3–25
   2.16. mf_enable_it() .................................................... 3–26
   2.17. mf_read_input_a() ............................................... 3–27
   2.18. mf_read_input_b() ............................................... 3–28
   2.19. mf_write_output_a() ............................................. 3–29
   2.20. mf_write_output_b() ............................................. 3–30
   2.21. mf_redundancy() ................................................. 3–31
CHAPTER 4 – INSTALLATION

1. DESCRIPTION ................................................................. 4–1

2. COMPILING OPTIONS ...................................................... 4–2
   2.1. Code Size ............................................................. 4–3
   2.2. Function Execution Times ......................................... 4–4
       2.2.1. PC equipped with a Pentium 120 MHz .................... 4–4
       2.2.2. Board equipped with a 8051 12 MHz ...................... 4–4

APPENDIX A – PROTOCOL PARAMETER SETTING

APPENDIX B – EXAMPLE OF SYSTEM CONFIGURATION AND APPLICATION

1. CONFIGURATION .......................................................... B–1
   1.1. Hardware and network configuration .......................... B–1
   1.2. Communication variable configuration ....................... B–2
   1.3. Messaging configuration ........................................... B–2
   1.4. Port configuration ................................................... B–2
   1.5. Interrupt configuration ............................................. B–2
   1.6. Synchronization identifier configuration ..................... B–2
   1.7. Identification variable configuration ......................... B–3

2. APPLICATION .............................................................. B–4

3. USER APPLICATION PROGRAM ......................................... B–5

4. CC165 INITIALIZATION ON A 16–BIT PC ............................ B–10

APPENDIX C – EXAMPLE OF SYSTEM CONFIGURATION AND APPLICATION USING THE NEW FUNCTIONS

1. CONFIGURATION TO BE PROCESSED ..................................... C–1
   1.1. Hardware and network configuration ......................... C–1
   1.2. Communication variable configuration ....................... C–2
   1.3. Messaging configuration ........................................... C–2
   1.4. Port configuration ................................................... C–3
   1.5. Interrupt configuration ............................................. C–3
   1.6. Synchronization identifier configuration ..................... C–3
   1.7. Identification variable configuration ......................... C–3

2. APPLICATION .............................................................. C–4

3. REDUNDANCY APPLICATION EXAMPLE ............................... C–9
1. OBJECTIVES

MICROFIP HANDLER is a software based on the WorldFIP technology developed by ALSTOM and called FIPWARE. It is composed of chips, software and tools. MICROFIP HANDLER runs on a microcontroller to easily access the MICROFIP chip. The User Reference Manual describes the library of available functions and provides additional use information for:

- compiling an application for a particular device,
- initializing a subscriber and configuring network exchanges,
- writing and reading the communication variables,
- sending and receiving messages.

The MICROFIP HANDLER is reserved for developers with WorldFIP knowledge who interface devices to the WorldFIP network using the MICROFIP chip set.
2. MICROFIP OVERVIEW

The MICROFIP chip runs the WorldFIP protocol supporting:

- Three standard speeds: 31.25 kbits/s, 1 Mbit/s and 2.5 Mbits/s.
- Different types of Medium Attachment Units (FIELDRIVE, CREOL, etc.).
- IEC or UTE frame delimiters and FCS.
- Up to 4 consumed and 4 produced variables with a total allocation capacity of 120 bytes.
- The variable size organised from 1 to 15 blocks of 8 bytes.
- Management of refreshment and promptness statuses.
- Identifiers from 0x00xy to 0x07xy, with xy as the subscriber number defined between 00 to 0xff.
- One identifier on a fully configurable consumed variable.
- One interrupt triggerable on a specific identifier.
- Up to 128 bytes of data including addresses for messaging.
- Message transmission with acknowledgement and retry.
- Message transmission without acknowledgement.
- An interrupt is attached to one produced variable (0x06xy) and to all the consuming variables (0x01xy, 0x03xy, 0x05xy, 0x07xy), and to the transmission and reception of messages.
- Aperiodic message request available on var 6 production.
- Two parallel input/output ports PIA and PIB with filtering option.
- Message reception in point–to–point or broadcast mode.
- One–line TO (time–out) parameter configuration.
- Possibility of purging messages to send.
- Resetting of line driver 1 or line driver 2 or both.
- Presence and identification variable production.
- Choice of starting up channels on reception of a variable or after a time delay of a second, or of never starting up the channels.
- Medium redundancy validation function.
1. INTRODUCTION

The MICROFIP HANDLER software is designed for use on the MICROFIP component. It is built as a library of functions or primitives to be linked with the user application and to be run on the MICROFIP component host processor. It transforms the equipment on which it is integrated into a WorldFIP subscriber that is implemented to only run as a station.

All the functions used to check the integrity of the subscriber configuration data are run during start-up. Checks are also run by functions called during run time. The parameters used in the functions can be checked in every function.

This library is written in C ANSI in order to be usable in the most environments possible. This library is also strictly independent of any context of use and no hypothesis is made on the execution model of the user application.
2. FUNCTION DEFINITIONS

The MICROFIP HANDLER software offers the following functions:

- **MANAGEMENT FUNCTION**
  This function is in charge of register initialization, and resetting and starting the subscriber.

- **INPUT/OUTPUT MANAGEMENT**
  This function is in charge of writing the output port and reading the input port.

- **COMMUNICATION VARIABLE MANAGEMENT**
  This function is in charge of configuring, writing and reading the communication variables.

- **MESSAGE MANAGEMENT**
  This function is in charge of configuring writing and reading the messages and the message send purge.

- **EVENT PROCESSING**
  This function is in charge of configuring the interrupts, reading the interrupt register, and disabling and enabling the interrupts.

- **REDUNDANCY VALIDATION FUNCTION**
  This function selects the best channel for reception when the basic mechanism included in the chip (selecting the first frame received) provides frames with an integrity problem (fragments, bad CRC, etc.). The reception quality of the current reception is evaluated during a tunable time period (T1) and the decision to force the reception of the other channel will be taken in accordance with a fault frame threshold. Testing re-insertion of the faulty channel is attempted with a longer tunable time period (T2). The use of this function requires specific hardware wiring between the MICROFIP chip and FIELDRIVE line transceiver (see also ALS 50280 MICROFIP User Reference Manual).
1. INTRODUCTION

The standard description for each primitive consists of the following items:

- **SYNOPTIC:** primitive definition
- **SYNTAX:** C format name and form of the primitive
- **DESCRIPTION:** detailed definition of the primitive
- **PARAMETER:** primitive input parameters definition
- **RESULT:** list of possible detected errors
2. LIST OF PRIMITIVES TO BE CALLED BY THE USER

The following primitives are enabled with the WITH–NEW–MICROFIP NO option:

<table>
<thead>
<tr>
<th>NAME</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>mf_initialize_network</td>
<td>component register initialization</td>
</tr>
<tr>
<td>mf_read_interrupt_register</td>
<td>interrupt register reading</td>
</tr>
<tr>
<td>mf_read_event</td>
<td>event reading</td>
</tr>
<tr>
<td>mf_var_read_loc</td>
<td>variable reading</td>
</tr>
<tr>
<td>mf_var_write_loc</td>
<td>variable writing</td>
</tr>
<tr>
<td>mf_read_message</td>
<td>message reading</td>
</tr>
<tr>
<td>mf_send_message</td>
<td>message sending</td>
</tr>
<tr>
<td>mf_disable_it</td>
<td>interrupt disabling</td>
</tr>
<tr>
<td>mf_enable_it</td>
<td>interrupt enabling</td>
</tr>
<tr>
<td>mf_read_input_A</td>
<td>input port PIA reading</td>
</tr>
<tr>
<td>mf_read_input_B</td>
<td>input port PIB reading</td>
</tr>
<tr>
<td>mf_write_output_A</td>
<td>output port PIA writing</td>
</tr>
<tr>
<td>mf_write_output_B</td>
<td>output port PIB writing</td>
</tr>
</tbody>
</table>
The following primitives are enabled with the WITH–NEW–MICROFIP YES option:

<table>
<thead>
<tr>
<th>NAME</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>mf_new_initialize_network</td>
<td>component register initialization except for the registers associated with the variables</td>
</tr>
<tr>
<td>mf_mps_var_conf</td>
<td>variable configuration</td>
</tr>
<tr>
<td>mf_read_interrupt_register</td>
<td>interrupt register reading</td>
</tr>
<tr>
<td>mf_read_event</td>
<td>event reading</td>
</tr>
<tr>
<td>mf_new_read_event</td>
<td>event reading</td>
</tr>
<tr>
<td>mf_var_read_loc</td>
<td>variable reading</td>
</tr>
<tr>
<td>mf_var_write_loc</td>
<td>variable writing</td>
</tr>
<tr>
<td>mf_read_message</td>
<td>message reading</td>
</tr>
<tr>
<td>mf_send_message</td>
<td>message sending</td>
</tr>
<tr>
<td>mf_purge_send_message</td>
<td>message purging</td>
</tr>
<tr>
<td>mf_send_message_after_purge</td>
<td>message sending</td>
</tr>
<tr>
<td>mf_reset_line_driver².</td>
<td>line driver resetting</td>
</tr>
<tr>
<td>mf_change_time_out_value</td>
<td>T0 parameter changing</td>
</tr>
<tr>
<td>mf_disable_it</td>
<td>interrupt disabling</td>
</tr>
<tr>
<td>mf_enable_it</td>
<td>interrupt enabling</td>
</tr>
<tr>
<td>mf_read_input_A</td>
<td>input port PIA reading</td>
</tr>
<tr>
<td>mf_read_input_B</td>
<td>input port PIB reading</td>
</tr>
<tr>
<td>mf_write_output_A².</td>
<td>output port PIA writing</td>
</tr>
<tr>
<td>mf_write_output_B</td>
<td>output port PIB writing</td>
</tr>
<tr>
<td>mf_redundancy¹.</td>
<td>medium redundancy validation</td>
</tr>
</tbody>
</table>

1. mf_redundancy requires the hardware implementation described in the MICROFIP User Reference Manual ALS 50280. The implementation of the mf_redundancy primitive invalidates the use of the mf_reset_line_driver and mf_write_output_A primitives.

2. You can use the mf_write_output_A and mf_reset_line_driver primitives if mf_redundancy is not selected (WITH_REDUndANCY=NO) in the compiling options. In this case RST1 and RST2 respectively reset for FIELDRI eve1 and FIELDRI eve2 should be directly connected to RST1N and RST2N MICROFIP pins.
2.1.  

mf_initialize_network()

SYNOPTIC:

Global initialization function of the MICROFIP chip.

SYNTAX:

```
unsigned short mf_initialize_network (
    MF_CONFIGURATION *User_configuration,
    MF_IDENTIFICATION *User_identification);
```

DESCRIPTION:

This function allows you to create the configuration of the MICROFIP HANDLER according to the specific needs of a subscriber. It has to be the first function called in the user application program. The input parameters are checked followed by initialization of the MICROFIP registers and start-up of the component and network activities.

PARAMETERS:

- **User_configuration**: user variable of MF_CONFIGURATION type which contains the MICROFIP parameter values to be assigned. These parameters depend on the required network configuration. They are defined as follows:

```c
typedef struct {
    unsigned char KPHYADR;
    MEMOIRE_MF *volatile Adr_Base;
    unsigned char Option_for_Phyadr;
    unsigned char Number_Of_Retries;
    unsigned char MAU_Type;
    unsigned char Standard_type;
    unsigned char Speed;
    unsigned char Turnaround_Time;
    unsigned char T0;
    unsigned char SEL_IDG;
    unsigned char ENABLE_IT;
    unsigned char Promptness_BankA;
    unsigned char Promptness_BankB;
    unsigned char Refreshment_BankA;
    unsigned char Refreshment_BankB;
    unsigned char Input_Ports_filter;
    unsigned short Global_ID;
    unsigned short Synchro_ID;
    unsigned char Num_Segment;
    unsigned char Quartz;
    unsigned short Var_Sizes[8];
} MF_CONFIGURATION;
```
KPHYADR : subscriber number value in [0:255]

Adr_Base : MICROFIP address in host processor addressing space

Option_for_Phyadr : value = 0 → subscriber number in KPHYADR
  1 → subscriber number acquired on the dedicated MICROFIP ports
  other → error

Number_Of_Retries : value = 0 → no retry for acknowledged messaging
  1 → one retry for acknowledged messaging
  other → error

Turnaround_Time : value = 0 → mandatory value at 1 Mbit/s (10 µs)
  0 → mandatory value at 2.5 Mbits/s (4 µs)
  7 → mandatory value at 31.25 kbits/s (64 µs)
  other → error

MAU_Type : value = 0 → FIELDRIVE type of MAU
  1 → CREOL type of MAU
  other → error

Standard_type : value = 0 → UTE frame delimiters and FCS
  1 → IEC frame delimiters and FCS
  other → error

Speed : value = 0 → 31.25 kbits/s
  1 → 1 Mbit/s
  2 → 2.5 Mbits/s
  other → error

T0 : value = possible values in [0:7]
  other → error (see Appendix A)

SEL_IDG : value = 0 → var7 ID number physically allocated
  1 → var7 ID number globally allocated
  other → error
ENABLE_IT
: bit7 = 1 enables interrupt on var7 reception
: bit6 = 1 enables interrupt on var5 reception
: bit5 = 1 enables interrupt on var3 reception
: bit4 = 1 enables interrupt on var1 reception
: bit3 = 1 enables interrupt on message transmission
: bit2 = 1 enables interrupt on message reception
: bit1 = 1 enables interrupt on Synchro_ID reception
: bit0 = 1 enables interrupt on var6 transmission

Promptness_BankB : value = 0 → promptness period of 50 ms on var5
1 → promptness period of 250 ms on var5
2 → promptness period of 1 s on var5
3 → promptness period of 5 s on var5
other → error

Promptness_BankA : value = 0 → promptness period of 50 ms on var1,3,7
1 → promptness period of 250 ms on var1,3,7
2 → promptness period of 1 s on var1,3,7
3 → promptness period of 5 s on var1,3,7
other → error

Refreshment_BankB : value = 0 → refreshment period of 250 ms on var6
1 → infinite refreshment period on var6
other → error

Refreshment_BankA : value = 0 → refreshment period of 250 ms on var0,2,4
1 → infinite refreshment period on var0,2,4
other → error

Input_Ports_filter : value = 0 → no filter applied on the input port
1 → filter of 1 ms
2 → filter of 4 ms
3 → filter of 10 ms
other → error
Global_ID : identifier number assigned on var depending on sel_idg condition

Synchro_ID : synchronisation identifier number depending on ENABLE_IT bit

Num_Segment : messaging segment number in [0 to 128]

Quartz : value = 0 → 20 MHz quartz
          1 → 40 MHz quartz
          other → error

Var_Sizes[8] : size in bytes of the 8 variables in multiples of 8 bytes:

  - a variable has a maximum length of 120 bytes,
  - the 8 variables have a maximum length of 120 bytes,
  - if a variable does not exist the length is 0.

- User_identification: user variable of MF_IDENTIFICATION type which contains the subscriber identification. This identification variable will be produced after execution of the mf_initialize_network function when receiving the 0X10XY identifier (where XY is the station number between 00 and 255).

Description of the MF_IDENTIFICATION type

typedef struct   {
    unsigned char Profile;
    unsigned char Class;
    unsigned char Constructor_h
    unsigned char Constructor_l;
    unsigned char Model_h;
    unsigned char Model_l;
    unsigned char Version;
    unsigned char User;
 }   MF_IDENTIFICATION;

To define Profile and Class see WorldFIP interoperability guides.
Constructor_h and Constructor_l are given by WorldFIP.
Model_h and Model_l are defined by the constructor.
Version is the product version.
User has to be defined by the user.
RESULT:

The `mf_initialize_network` function returns a report if requested by the user; it is an unsigned short char (16 bits), with each bit representing an error if set to 1 as summarised below:

<table>
<thead>
<tr>
<th>BIT SET TO 1</th>
<th>MEANING</th>
<th>HELP</th>
</tr>
</thead>
<tbody>
<tr>
<td>no bit set to 1</td>
<td>OK</td>
<td>no error</td>
</tr>
<tr>
<td>bit0</td>
<td>ERR_OPT_PHYADR</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit1</td>
<td>ERR_CONF_NUM_ACK</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit2</td>
<td>ERR_CONF_SEL_TRP</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit3</td>
<td>ERR_CONF_SEL_COL</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit4</td>
<td>ERR_CONF_SEL_CEI</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit5</td>
<td>ERR_CONF_SEL_FRQ</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit6</td>
<td>ERR_CONF_SEL_IDG</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit7</td>
<td>ERR_CONF_SEL_TOT</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit8</td>
<td>ERR_MPS_SEL_BPR</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit9</td>
<td>ERR_MPS_SEL_APR</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit10</td>
<td>ERR_MPS_SEL_BRA</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit11</td>
<td>ERR_MPS_SEL_ARA</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit12</td>
<td>ERR_MPS_SEL_IFI</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit13</td>
<td>ERR_BASIC_SEL_QTZ</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit14</td>
<td>ERR_NO_SEGMENT</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit15</td>
<td>ERR_SIZE_VAR</td>
<td>parameter outside range</td>
</tr>
</tbody>
</table>
2.2. \texttt{mf\_new\_initialize\_network()}

\textbf{SYNOPTIC:}

Global initialization function of the MICROFIP chip.

\textbf{SYNTAX:}

\begin{verbatim}
unsigned short mf_new_initialize_network
    MF_NEW_CONFIGURATION *User_new_configuration,
    MF_IDENTIFICATION *User_identification);
\end{verbatim}

\textbf{DESCRIPTION:}

This function allows you to create the configuration of the MICROFIP HANDLER according to the specific needs of a subscriber. It has to be the first function called in the user application program. The input parameters are checked, followed by initialization of the MICROFIP registers and start-up of the required component and the network activities.

\textbf{PARAMETERS:}

\texttt{User\_new\_configuration:} user variable of \texttt{MF\_NEW\_CONFIGURATION} type which contains the MICROFIP parameter values to be assigned. These parameters depend on the required network configuration. They are defined as follows:

\begin{verbatim}
typedef struct { unsigned char KPHYADR;
    MEMOIRE_MF *volatile Adr_Base;
    unsigned char Option_for_Phyadr;
    unsigned char Number_Of_Retries;
    unsigned char MAU_Type;
    unsigned char Standard_type;
    unsigned char Startup_Channel;
    unsigned char Msg_rec_Mode;
    unsigned char Speed;
    unsigned char Turnaround_Time;
    unsigned char T0;
    unsigned char ENABLE_IT;
    unsigned char Input_Ports_filter;
    unsigned char Num_Segment;
    unsigned char Quartz;
    unsigned short Tempo_Redundancy_T1;
    unsigned short Tempo_Redundancy_T2;
} MF_NEW_CONFIGURATION;
\end{verbatim}

\textbf{Note}

\texttt{Tempo\_Redundancy\_T1} and \texttt{Tempo\_Redundancy\_T2} depend on a compilation option.
KPHYADR : subscriber number value between 0 and 255

*volatile Adr_Base : MICROFIP address in host processor addressing space

Option_for_Phyadr : value =
                   0 → subscriber number in KPHYADR
                   1 → subscriber number acquired on the MICROFIP
dedicated ports
                   other → error

Number_Of_Retries : value =
                     0 → no retry for acknowledged messaging
                     1 → one retry for acknowledged messaging
                     other → error

MAU_Type : value =
            0 → FIELDRIVE type of MAU
            1 → CREOL type of MAU
            other → error

Standard_type : value =
                1 → IEC frame delimiters and FCS
                0 → UTE frame delimiters and FCS
                other → error

Startup_Channel : value =
                  0 → no change
                  1 → on presence variable
                  2 → on presence variable and/or one–second time–out
                  3 → not active

Msg_rec_Mode : value =
               0 → no change
               1 → message reception disabled
               2 → point–to–point message reception enabled
               3 → broadcast and point–to–point message reception enabled

Speed : value =
        0 → 31.25 kbits/s
        1 → 1 Mbit/s
        2 → 2.5 Mbit/s
        other → error

Turnaround_Time : value =
                   0 → mandatory value at 1 Mbit/s (10 µs)
                   3 → mandatory value at 2.5 Mbits/s (16 µs)
                   1 → mandatory value at 31.25 kbits/s (640 µs)
                   other → error (see Appendix A)

T0 : value =
     2 → 4160 µs at 31.25 kbits/s
     6 → 6720 µs at 31.25 kbits/s
     3 → 150 µs at 1 Mbit/s
     7 → 250 µs at 1 Mbit/s
     7 → 100 µs at 2.5 Mbit/s
     other → error (see Appendix A)

ENABLE_IT : bit7 = 1 → enables interrupt on var7 reception
           : bit6 = 1 → enables interrupt on var5 reception
Description of user interface primitives

: bit5 = 1 → enables interrupt on var3 reception
: bit4 = 1 → enables interrupt on var1 reception
: bit3 = 1 → enables interrupt on message transmission
: bit2 = 1 → enables interrupt on message reception
: bit1 = 1 → enables interrupt on Synchro_ID reception
: bit0 = 1 → enables interrupt on var6 transmission

Input_Ports_filter : value =
  0 → no filter applied on the input port
  1 → filter of 1 ms
  2 → filter of 4 ms
  3 → filter of 10 ms
  other → error

Num_Segment : messaging segment number between 0 and 128

Quartz : value =
  0 → 20 MHz quartz
  1 → 40 MHz quartz
  other → error

Tempo_Redundancy_T1 : value ≥
  300 ms
  other → error

Tempo_Redundancy_T2 : value ≥
  2 mn
  other → error

RESULT:

The mf_new_initialize_network function returns a report if requested by the user; it is an unsigned short char (16 bits). Each bit represents an error if it is set to 1 as summarized below:

<table>
<thead>
<tr>
<th>BIT SET TO 1</th>
<th>MEANING</th>
<th>HELP</th>
</tr>
</thead>
<tbody>
<tr>
<td>no bit set to 1</td>
<td>OK</td>
<td>no error</td>
</tr>
<tr>
<td>bit0</td>
<td>ERR_OPT_PHYADR</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit1</td>
<td>ERR_CONFA_NUM_ACK</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit2</td>
<td>ERR_CONFC_SEL_TRP</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit3</td>
<td>ERR_CONFC_SEL_COL</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit4</td>
<td>ERR_CONFC_SEL_CEi</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit5</td>
<td>ERR_CONFC_SEL_FRQ</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit6</td>
<td>ERR_CONFD_SEL_TOT</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit7</td>
<td>ERR_CONFA_STU_CHL</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit8</td>
<td>ERR_CONFA_MSG_REC</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit10</td>
<td>ERR_COMPOSANT_PAS_VU</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit11</td>
<td>ERR_CONF_REUNDANCY</td>
<td>one or two parameter values too small</td>
</tr>
<tr>
<td>bit12</td>
<td>ERR_MPSPR_SEL_IFI</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit13</td>
<td>ERR_BASIC_SEL_QTZ</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit14</td>
<td>ERR_NO_SEGMENT</td>
<td>parameter outside range</td>
</tr>
</tbody>
</table>
2.3. mf_mps_var_conf()

SYNOPTIC:
Global initialization function of the MICROFIP chip.

SYNTAX:

```c
unsigned short mf_mps_var_conf ( 
    MF_VAR_CONF *User_var_conf);
```

DESCRIPTION:
This function allows you to create the variable configuration of the MICROFIP HANDLER according to the specific needs of a subscriber. After the input parameter check, the MICROFIP registers are initialized.

PARAMETERS:

User_var_conf: user variable of MF_VAR_CONF type which contains the MICROFIP parameter values to be assigned. You should choose these parameters depending on the network configuration needed. They are defined as follows.

```c
typedef struct { 
    unsigned char SEL_IDG; 
    unsigned char Promptness_BankA; 
    unsigned char Promptness_BankB; 
    unsigned char Refreshment_BankA; 
    unsigned char Refreshment_BankB; 
    unsigned short Global_ID; 
    unsigned short Synchro_ID; 
    unsigned short Var_Sizes[8]; 
} MF_VAR_CONF;
```

SEL_IDG : value = 0 → var7 ID number physically allocated
1 → var7 ID number globally allocated
other → error

Promptness_BankB : value = 0 → promptness period of 50 ms on var5
1 → promptness period of 250 ms on var5
2 → promptness period of 1 s on var5
3 → promptness period of 5 s on var5
other → error

Promptness_BankA : value = 0 → promptness period of 50 ms on var1,3,7
1 → promptness period of 250 ms on var1,3,7
2 → promptness period of 1 s on var1,3,7
3 → promptness period of 5 s on var1,3,7
other → error
Refreshment_BankB : value = 0 → refreshment period of 250 ms on var6
1 → infinite refreshment period on var6
other → error

Refreshment_BankA : value = 0 → refreshment period of 250 ms on var0,2,4
1 → infinite refreshment period on var0,2,4
other → error

Global_ID : identifier number assigned on var7
depending on SEL_IDG condition

Synchro_ID : synchronism identifier number
depending on ENABLE_IT bit1

Var_Sizes[8] : size in bytes of the 8 variables in multiples of 8 bytes
• a variable has a maximum length of 120 bytes
• the 8 variables have a maximum length of 120 bytes
• if a variable does not exist the length is 0

RESULT:
The `mf_mps_var_conf` function returns a report if requested by the user; it is an unsigned short (16 bits). Each bit represents an error if it is set to 1 as summarized below:

<table>
<thead>
<tr>
<th>BIT SET TO 1</th>
<th>MEANING</th>
<th>HELP</th>
</tr>
</thead>
<tbody>
<tr>
<td>no bit set to 1</td>
<td>OK</td>
<td>no error</td>
</tr>
<tr>
<td>bit0</td>
<td>ERR_CONFID_SEL_IDG</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit1</td>
<td>ERR_MPSPR_SEL_BPR</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit2</td>
<td>ERR_MPSPR_SEL_APR</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit3</td>
<td>ERR_MPSPR_SEL_BRA</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit4</td>
<td>ERR_MPSPR_SEL_ARA</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit5</td>
<td>ERR_SIZE_VAR</td>
<td>parameter outside range</td>
</tr>
</tbody>
</table>
2.4. mf_read_interrupt_register()

SYNOPTIC:
Reading the interrupt register of MICROFIP.

SYNTAX:

unsigned char mf_read_interrupt_register (void);

DESCRIPTION:
This function allows you to get the content of the MICROFIP interrupt register. This content gives you information on the network activities such as reception and transmission of variables, reception and transmission of messages, and reception of synchronisation ID. This function is also used for medium redundancy validation to detect activity on the network.

PARAMETER:
No input parameter.

RESULT:
The mf_read_interrupt_register function returns to the user a report where each bit set to 1 has the following meaning:

<table>
<thead>
<tr>
<th>bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>var7 reception</td>
</tr>
<tr>
<td>6</td>
<td>var5 reception</td>
</tr>
<tr>
<td>5</td>
<td>var3 reception</td>
</tr>
<tr>
<td>4</td>
<td>var1 reception</td>
</tr>
<tr>
<td>3</td>
<td>message reception</td>
</tr>
<tr>
<td>2</td>
<td>message transmission</td>
</tr>
<tr>
<td>1</td>
<td>synchro reception</td>
</tr>
<tr>
<td>0</td>
<td>var6 transmission</td>
</tr>
</tbody>
</table>
2.5. mf_read_event()

SYNOPTIC:
Reading the event register of the MICROFIP.

SYNTAX:

```c
unsigned char mf_read_event(void);
```

DESCRIPTION:

This function allows you to get the content of the MICROFIP event register. This content gives you information on the hardware state when the FIELDRIVE option is chosen.

PARAMETER:

No input parameter.

RESULT:

The `mf_read_event` function returns a bit set with the following meaning:

- `bit7 = 1`: watchdog error channel 2
- `bit6 = 1`: watchdog error channel 1
- `bit5`: reserved
- `bit4`: reserved
- `bit3`: reserved
- `bit2`: reserved
- `bit1`: reserved
- `bit0`: reserved
2.6. **mf_new_read_event()**

**SYNOPTIC:**

Reading the event register of the MICROFIP.

**SYNTAX:**

```c
unsigned char mf_new_read_event(void);
```

**DESCRIPTION:**

This function allows you to get the content of the MICROFIP event register. This content gives you information on the hardware state when the FIELDRIVE option is chosen.

**PARAMETER:**

No input parameter

**RESULT:**

The `mf_new_read_event` function returns a bit set with the following meaning:

- `bit7 = 1` watchdog error channel 2
- `bit6 = 1` watchdog error channel 1
- `bit5 = 1` error traced on receiver activities
- `bit4 = 1` overflow traced on messaging receiver activities
- `bit3 = 1` lack of acknowledgement
- `bit2 = 1` ACK– received
- `bit2 = 0` ACK+ received or non acknowledged messaging activity
- `bit1 = 1` messaging transmitter buffer empty
- `bit1 = 0` messaging transmitter buffer occupied
- `bit0 = 1` messaging receiver buffer contains a valid message
- `bit0 = 0` messaging receiver buffer contains no valid message
2.7. mf_var_read_loc()

SYNOPTIC:
Reading of a communication variable.

SYNTAX:
unsigned char mf_var_read_loc (
    unsigned short variable_number,
    MF_VAR_DATA *storage_buffer);

DESCRIPTION:
This function allows you to read the value of a received communication variable.

PARAMETER:

variable_number : the relative variable number in the MICROFIP, which is [1 or 3 or 5 or 7]

*storage_buffer : a pointer to the MF_VAR_DATA type structure defined as following.

typedef struct { unsigned char Buffer[64] ;
} MF_VAR_DATA ;

RESULT:
The mf_var_read_loc function returns a report to the user. This report is an unsigned char where each bit set to 1 represents an error resulting from MICROFIP parameter validity and variable statuses.

<table>
<thead>
<tr>
<th>BIT SET TO 1</th>
<th>MEANING</th>
<th>HELP</th>
</tr>
</thead>
<tbody>
<tr>
<td>no bit set to 1</td>
<td>OK</td>
<td>no error</td>
</tr>
<tr>
<td>bit0</td>
<td>ERR_VAR_NUMBER_R</td>
<td>variable_number outside range</td>
</tr>
<tr>
<td>bit1</td>
<td>ERR_VAR_SIZE_R</td>
<td>non existing variable</td>
</tr>
<tr>
<td>bit2</td>
<td>ERR_VAR_STATUS_R</td>
<td>status associated with the false variable</td>
</tr>
</tbody>
</table>
2.8. mf_var_write_loc()

SYNOPTIC:
Writing of a communication variable.

SYNTAX:
unsigned char mf_var_write_loc (
    unsigned short variable_number,
    unsigned char *data_buffer);

DESCRIPTION:
This function allows you to write the value of a produced communication variable.

PARAMETER:
variable_number : the relative variable number in the MICROFIP, which is [0 or 2 or 4 or 6].
*data_buffer : pointer to the variable data.

RESULT:
The mf_var_write_loc function returns a report to the user. This report is an unsigned char where each bit set to 1 represents an error resulting from MICROFIP parameter validity and variable statuses.

<table>
<thead>
<tr>
<th>BIT SET TO 1</th>
<th>MEANING</th>
<th>HELP</th>
</tr>
</thead>
<tbody>
<tr>
<td>no bit set to 1</td>
<td>OK</td>
<td>no error</td>
</tr>
<tr>
<td>bit0</td>
<td>ERR_VAR_NUMBER_W</td>
<td>variable_number outside range</td>
</tr>
<tr>
<td>bit1</td>
<td>ERR_VAR_SIZE_W</td>
<td>variable not configured</td>
</tr>
<tr>
<td>bit2</td>
<td>ERR_BUFFER_BUSY_W</td>
<td>buffer busy</td>
</tr>
</tbody>
</table>
2.9. mf_read_message()

SYNOPTIC:

Reading of a received message.

SYNTAX:

```c
unsigned char mf_read_message (
    MF_MSG_RECEIVED *ptr_msg_receive);
```

DESCRIPTION:

This function allows you to read a message from the MICROFIP memory.

PARAMETER:

*ptr_msg_receive : a pointer to the MF_MSG_RECEIVED type defined as follows:

```c
typedef struct { unsigned short Size ;
    unsigned char DATA_BUFFER[128] ;
} MF_MSG_RECEIVED ;
```

RESULT:

The mf_read_message function returns a report to the user. This report is an unsigned char where each bit set to 1 represents an error resulting from MICROFIP parameter and message availability:

- If an error is detected the message is not read.
- If no error is detected the report is 0 and the structure pointed to by *ptr_msg_receive is updated with the size of the received message, including the six bytes of messaging addresses and the data read in the buffer.

<table>
<thead>
<tr>
<th>BIT SET TO 1</th>
<th>MEANING</th>
<th>HELP</th>
</tr>
</thead>
<tbody>
<tr>
<td>no bit set to 1</td>
<td>OK</td>
<td>no error</td>
</tr>
<tr>
<td>bit0</td>
<td>ERR_MSG_NOT_READY_R</td>
<td>no message ready to be read</td>
</tr>
</tbody>
</table>
2.10. mf_send_message()

SYNOPTIC:
Writing of a message to be transmitted.

SYNTAX:

```c
unsigned char mf_send_message (
    unsigned short Size,
    unsigned char sel_ack,
    unsigned char *data_buffer);
```

DESCRIPTION:

This function allows you to write the message in the MICROFIP memory, specifying the size and the type of exchange requested (with or without acknowledgement).

PARAMETER:

- **Size**: unsigned short indicating the size of the message, including the six bytes of messaging addresses (the maximum number of user data is 122 bytes).
- **sel_ack**: type of messaging used to transmit (with or without acknowledgement).
- ***data_buffer**: pointer to the message data.

RESULT:

The `mf_send_message` function returns a report to the user. This report is an unsigned char where each bit set to 1 represents an error resulting from MICROFIP parameter validity and messaging availability.

<table>
<thead>
<tr>
<th>BIT SET TO 1</th>
<th>MEANING</th>
<th>HELP</th>
</tr>
</thead>
<tbody>
<tr>
<td>no bit set to 1</td>
<td>OK</td>
<td>no error</td>
</tr>
<tr>
<td>bit0</td>
<td>ERR_CONFA_SEL_ACK_S</td>
<td>messaging type outside range</td>
</tr>
<tr>
<td>bit1</td>
<td>ERR_MSG_SIZE_S</td>
<td>message size outside range</td>
</tr>
<tr>
<td>bit2</td>
<td>ERR_MSG_QUEUE_BUSY_S</td>
<td>message queue busy</td>
</tr>
<tr>
<td>bit3</td>
<td>ERR_MSG_ADR</td>
<td>address error</td>
</tr>
</tbody>
</table>
2.11. mf_purge_send_message()

SYNOPTIC:
Purging of a message to be transmitted.

SYNTAX:

unsigned char mf_purge_send_message ()

DESCRIPTION:
This function allows you to purge the message in the MICROFIP memory.

PARAMETER:
No input parameter

RESULT:
The mf_purge_send_message function returns a report to the user. This report is an unsigned char where each bit set to 1 represents an error resulting from MICROFIP parameter validity and messaging availability.

<table>
<thead>
<tr>
<th>BIT SET TO 1</th>
<th>MEANING</th>
<th>HELP</th>
</tr>
</thead>
<tbody>
<tr>
<td>no bit set to 1</td>
<td>OK</td>
<td>no error</td>
</tr>
<tr>
<td>bit0</td>
<td>ERR_NO_MSG_TO_PURGE</td>
<td>messaging type outside range</td>
</tr>
</tbody>
</table>
2.12. mf_send_message_after_purge()

SYNOPTIC:
Writing of a message to be transmitted when a mf_purge_send_message function has been executed.

SYNTAX:

```c
unsigned char mf_send_message_after_purge ( 
    unsigned short Size, 
    unsigned char sel_ack, 
    unsigned char *data_buffer );
```

DESCRIPTION:
This function allows you to write the message in the MICROFIP memory and to specify the size and the type of the requested exchange (with or without acknowledgement).

PARAMETER:

- **Size**: unsigned short that indicates the size of the message, including the 6 bytes of messaging addresses (the maximum number of user data is 122 bytes).
- **sel_ack**: type of messaging used to transmit (with or without acknowledgement).
- **data_buffer**: pointer to the message data.

RESULT:
The mf_send_message_after_purge function returns a report to the user. This report is an unsigned char where each bit set to 1 represents an error resulting from MICROFIP parameter validity and messaging availability.

<table>
<thead>
<tr>
<th>BIT SET TO 1</th>
<th>MEANING</th>
<th>HELP</th>
</tr>
</thead>
<tbody>
<tr>
<td>no bit set to 1</td>
<td>OK</td>
<td>no error</td>
</tr>
<tr>
<td>bit0</td>
<td>ERR_CONF_SEL_ACK_S</td>
<td>messaging type outside range</td>
</tr>
<tr>
<td>bit1</td>
<td>ERR_MSG_SIZE_S</td>
<td>message size outside range</td>
</tr>
<tr>
<td>bit3</td>
<td>ERR_MSG_ADR</td>
<td>address error</td>
</tr>
</tbody>
</table>
2.13. mf_reset_line_driver()

SYNOPSIS:
Resetting of line driver 1 or line driver 2 or both.

SYNTAX:

```c
unsigned char mf_reset_line_driver ( 
    unsigned char line);
```

DESCRIPTION:
This function allows you to reset a line driver or both line drivers when a watchdog error channel has been detected in MICROFIP.

PARAMETER:

- `line`:
  - value = 1 → resets line driver 1
  - value = 2 → resets line driver 2
  - value = 3 → resets line driver 1 and 2

RESULT:
The `mf_reset_line_driver` function returns a report to the user. This report is an unsigned char where each bit set to 1 represents an error resulting from MICROFIP parameter validity and messaging availability.

<table>
<thead>
<tr>
<th>BIT SET TO 1</th>
<th>MEANING</th>
<th>HELP</th>
</tr>
</thead>
<tbody>
<tr>
<td>no bit set 1</td>
<td>OK</td>
<td>no error</td>
</tr>
<tr>
<td>bit0</td>
<td>ERR_LINE_DRIVER_NB</td>
<td>messaging type outside range</td>
</tr>
</tbody>
</table>

Note
This function is not usable if the WITH_REDUNDANCY compiling option is selected.
2.14. mf_change_time_out_value()

SYNOPTIC:
Changing of the time-out value in the MICROFIP chip online.

SYNTAX:

void mf_change_time_out_value ( LIST_CONF_T0_value );

DESCRIPTION:
This function allows you to change the value of the Time_Out parameter online in the MICROFIP chip register.

PARAMETER:

LIST_CONF_T0

value = 0 → default value of T0
value = 1 → value of T0 used in an extended network

The T0 values are defined according to the authorised values:

<table>
<thead>
<tr>
<th>Speed Value</th>
<th>Value</th>
<th>T0 (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.25 kbits/s</td>
<td>0</td>
<td>4160</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>6720</td>
</tr>
<tr>
<td>1 Mbit/s</td>
<td>0</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>250</td>
</tr>
<tr>
<td>2.5 Mbits/s</td>
<td>0</td>
<td>100</td>
</tr>
</tbody>
</table>

RESULT:

The mf_change_time_out_value function returns a report to the user. This report is an unsigned char where each bit set to 1 represents an error resulting from MICROFIP parameter validity and messaging availability.

<table>
<thead>
<tr>
<th>BIT SET TO 1</th>
<th>MEANING</th>
<th>HELP</th>
</tr>
</thead>
<tbody>
<tr>
<td>no bit set to 1</td>
<td>OK</td>
<td>no error</td>
</tr>
<tr>
<td>bit0</td>
<td>ERR_CHANGE_T0_1</td>
<td>parameter outside range</td>
</tr>
<tr>
<td>bit1</td>
<td>ERR_CHANGE_T0_2</td>
<td>speed value read in register HS</td>
</tr>
</tbody>
</table>
2.15. mf_disable_it()

SYNOPTIC:
Disabling interrupt function.

SYNTAX:

```c
void mf_disable_it (enum numero_it Num_IT);
```

DESCRIPTION:
This function allows you to disable an interrupt by specifying its number.

PARAMETER:

- `Num_IT` : interrupt to be disabled, defined as follows:

```c
class numero_it {
    it0 = 1,
    it1 = 2,
    it2 = 4,
    it3 = 8,
    it4 = 0x10,
    it5 = 0x20,
    it6 = 0x40,
    it7 = 0x80,
    it_all=0xFF
};
```

- `it0` disables it on var7 reception
- `it1` disables it on var5 reception
- `it2` disables it on var3 reception
- `it3` disables it on var1 reception
- `it4` disables it on message transmission
- `it5` disables it on message reception
- `it6` disables it on synchro reception
- `it7` disables it on var6 transmission

RESULT:

The `mf_disable_it` function does not return a report.
2.16. mf_enable_it()

SYNOPTIC:

Enabling interrupt function.

SYNTAX:

void mf_enable_it (enum numero_it Num_IT);

DESCRIPTION:

This function allows you to enable an interrupt by specifying its number.

PARAMETER:

Num_IT : interrupt to be enabled, defined as follows:

```
enum numero_it {
    it0 = 1,
    it1 = 2,
    it2 = 4,
    it3 = 8,
    it4 = 0x10,
    it5 = 0x20,
    it6 = 0x40,
    it7 = 0x80,
    it_all=0xFF
};
```

- it0 enables it on var7 reception
- it1 enables it on var5 reception
- it2 enables it on var3 reception
- it3 enables it on var1 reception
- it4 enables it on message transmission
- it5 enables it on message reception
- it6 enables it on synchro reception
- it7 enables it on var6 transmission

RESULT:

The mf_enable_it function does not return a report.
2.17. mf_read_input_a()

SYNOPTIC:
Reading the value of the input port PIA.

SYNTAX:

unsigned char mf_read_input_A (void);

DESCRIPTION:
This function allows you to read the value of the input port specified as PIA.

PARAMETER:
No parameter

RESULT:
The `mf_read_input_A` function returns to the user an unsigned char representing the value of the specified port.
2.18. mf_read_input_b()

SYNOPTIC:

Reading the value of the input port PIB.

SYNTAX:

unsigned char mf_read_input_b (void);

DESCRIPTION:

This function allows you to read the value of the input port specified as PIB.

PARAMETER:

No parameter

RESULT:

The mf_read_input_b function returns to the user an unsigned char representing the value of the specified port.
2.19. **mf_write_output_a()**

**SYNOPTIC:**

Writing of the output port PIA.

**SYNTAX:**

```c
void mf_write_output_A (unsigned char Output_Value);
```

**DESCRIPTION:**

This function allows you to write the content of the `Output_Value` variable in the output port PIA.

**PARAMETER:**

`Output_Value` : value to update the `Output_Port`.

**RESULT:**

No report

---

**Note**

This function is not usable if the `WITH_REDUNDANCY` compiling option is selected.
2.20. mf_write_output_b()

SYNOPTIC:

Writing of the output port PIB.

SYNTAX:

void mf_write_output_B (unsigned char Output_Value);

DESCRIPTION:

This function allows you to write the content of the Output_Value variable in the output port PIB.

PARAMETER:

Output_Value : value to update the Output_Port.

RESULT:

No report
2.21. mf_redundancy()

SYNOPTIC:
Channel management.

SYNTAX:

unsigned char mf_redundancy ( void );

DESCRIPTION:
This function manages channel redundancy only if the redundancy option is set to YES.

If yes, management is done in three parts; the first part takes place in the mf_new_initialize_network where channels 1 and 2 are set to valid. The second appears in the mf_read_interrupt_register function: any time an interrupt is read to 1 a status flag associated with activity is set to 1. (The mf_read_interrupt_register function has to be called periodically by the user application in pooling mode.) The third part is done inside the mf_redundancy function. It is compulsory to call this function every 10 ms in the user application.

If the status flag associated with activity is read to 0 for the Tempo_Reundancy_T1 delay, channel management processing is executed. The current state of the channels then leads to a new configuration defined as follows:

if channel 1 state is reset and channel 2 state is valid then reset channel 1 and validate channel 2.

else, validate channel 1 and reset channel 2

Each time redundancy processing leading to a new channel configuration is executed, the Tempo_redundancy_T2 delay is activated. When the delay ends the two channels are set to valid.

PARAMETER:
No input parameter.

RESULT:
The mf_redundancy function returns a report to the user. This report is an unsigned char representing the current states of the channels if different from 0xff.

<table>
<thead>
<tr>
<th>VALUES</th>
<th>MEANING</th>
<th>HELP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xff</td>
<td>NOK</td>
<td>Redundancy not available due to bad Tempo_Reundancy_T1 and/or Tempo_Reundancy_T2 parameter settings if controls option is NO</td>
</tr>
<tr>
<td>01</td>
<td>RESET CHANNEL 2 VALID CHANNEL 1</td>
<td></td>
</tr>
<tr>
<td>bit1</td>
<td>RESET CHANNEL 1 VALID CHANNEL 2</td>
<td></td>
</tr>
<tr>
<td>bit0</td>
<td>VALID CHANNEL 1 VALID CHANNEL 2</td>
<td></td>
</tr>
</tbody>
</table>
1. DESCRIPTION

The MICROFIP HANDLER contains the following files:

- `user_opt.h`: used to define compilation options,
- `mfhd.h`: function, variable, type and constant definitions,
- `mfhd.c`: MICROFIP HANDLER functions,
- `mftst1.c`: user program application example,
2. COMPILING OPTIONS

The compiling options to be defined in the `user_opt.h` file are as follows:

```c
#ifndef __user_opt_h
#define __user_opt_h
#define YES 1
#define NO 0
#define WITH_NEW_MICROFIP YES
#define WITH_CONTROLS YES
#define WITH_MESSAGING YES
#define WITH_IO_PORT YES
#define WITH_ENA_DIS_ITS YES
#define WITH_CHANGE_TO YES
#define WITH_REDUNDANCY YES
#define WITH_8051 NO
#define WITH_FIPIULIB NO
```

The user has to select the following options:

- **WITH_NEW_MICROFIP**
  - YES: with Vy 27 257 MICROFIP component
  - NO: with Vy 27 190 MICROFIP component

- **WITH_CONTROLS**
  - YES: with control on parameters
  - NO: without control (will reduce the code size)

- **WITH_MESSAGING**
  - YES: with the use of messaging services
  - NO: without messaging services

- **WITH_IO_PORT**
  - YES: with the use of I/O ports
  - NO: without use of I/O ports

- **WITH_ENA_DIS_ITS**
  - YES: with the use of interrupt management
  - NO: without interrupt management

- **WITH_CHANGE_TO**
  - YES: with the use of interrupt management
  - NO: without change of TO

- **WITH_8051**
  - YES: with the use of 8051 processor
  - NO: without use of 8051 processor

- **WITH_FIPIULIB**
  - YES: with FIPIULIB library compatibility
  - NO: without FIPIULIB library compatibility

- **WITH_REDUNDANCY**
  - YES: with the use of medium redundancy validation
  - NO: without medium redundancy validation
Note

If WITH_REDUNDANCY = YES, you have to call the mf_redundancy function every 10 ms and the mf_read_interrupt_register function at least once every T1 delay.

WITH_FIPIULIB = YES means that the users want to have the same user interface (same functions to call, with the same parameters) as for FIPIULIB.

This feature is executed by adding software to translate the MICROFIP HANDLER interface to FIPIULIB interface.

You will find further information on this software in: MICROFIP HANDLER – FIPIULIB OPTION V01 – oz REFERENCE MANUAL 1080403 – V0102. For more details and technical support please contact WorldFIP Organisation.

2.1. Code Size

The code size for the MICROFIP HANDLER object code is given for a configuration with BORLAND C compiler on PC and depending on the selected options:

- WITH_NEW_MICROFIP YES
  WITH_CONTROLS YES
  WITH_MESSAGING YES
  WITH_IO_PORT YES
  WITH_DIS_ENA_ITS YES
  WITH_CHANGE_TO YES
  WITH_REDUNDANCY YES
  WITH_8051 NO
  WITH_FIPIULIB NO
  → code size = 3820 bytes if speed optimization
  → code size = 3680 bytes if code optimization
  → data size = 78 bytes

- WITH_NEW_MICROFIP YES
  WITH_CONTROLS NO
  WITH_MESSAGING YES
  WITH_IO_PORT YES
  WITH_DIS_ENA_ITS YES
  WITH_CHANGE_TO YES
  WITH_REDUNDANCY YES
  WITH_8051 NO
  WITH_FIPIULIB NO
  → code size = 2773 bytes if speed optimization
  → code size = 2656 bytes if code optimization
  → data size = 78 bytes
2.2. Function Execution Times

2.2.1. PC equipped with a Pentium 120 MHz

mf_var_write_loc function duration with the option WITH_CONTROLS = YES

duration = 13 + (number_of_bytes * 0.854) µs

mf_send_message function duration with the option WITH_CONTROLS = YES

duration = 13 + (number_of_bytes * 0.862) µs

mf_var_read_loc function duration with the option WITH_CONTROLS = YES

duration = 14.8 + (number_of_bytes * 0.916) µs

mf_read_message function duration with the option WITH_CONTROLS = YES

duration = 12.68 + (number_of_bytes * 0.940) µs

2.2.2. Board equipped with a 8051 12 MHz

mf_var_write_loc function duration with the option WITH_CONTROLS = YES

duration = 700 + (number_of_bytes * 21.9) µs

mf_send_message function duration with the option WITH_CONTROLS = YES

duration = 717 + (number_of_bytes * 22) µs

mf_var_read_loc function duration with the option WITH_CONTROLS = YES

duration = 746 + (number_of_bytes * 21.9) µs

mf_read_message function duration with the option WITH_CONTROLS = YES

duration = 530 + (number_of_bytes * 22) µs
## Protocol parameter setting

### Turnaround_Time

<table>
<thead>
<tr>
<th>Parameter value</th>
<th>31.25 kbits/s</th>
<th>1 Mbit/s</th>
<th>2.5 Mbits/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reserved</td>
<td>10 µs</td>
<td>reserved</td>
</tr>
<tr>
<td>1</td>
<td>640 µs</td>
<td>reserved</td>
<td>reserved</td>
</tr>
<tr>
<td>2</td>
<td>reserved</td>
<td>reserved</td>
<td>reserved</td>
</tr>
<tr>
<td>3</td>
<td>reserved</td>
<td>reserved</td>
<td>16 µs</td>
</tr>
<tr>
<td>4</td>
<td>reserved</td>
<td>reserved</td>
<td>reserved</td>
</tr>
<tr>
<td>5</td>
<td>reserved</td>
<td>reserved</td>
<td>reserved</td>
</tr>
<tr>
<td>6</td>
<td>reserved</td>
<td>reserved</td>
<td>reserved</td>
</tr>
<tr>
<td>7</td>
<td>reserved</td>
<td>reserved</td>
<td>reserved</td>
</tr>
</tbody>
</table>

### T0: silence time

<table>
<thead>
<tr>
<th>Parameter value</th>
<th>31.25 kbits/s</th>
<th>1 Mbit/s</th>
<th>2.5 Mbits/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reserved</td>
<td>reserved</td>
<td>reserved</td>
</tr>
<tr>
<td>1</td>
<td>reserved</td>
<td>reserved</td>
<td>reserved</td>
</tr>
<tr>
<td>2</td>
<td>4160 µs</td>
<td>reserved</td>
<td>reserved</td>
</tr>
<tr>
<td>3</td>
<td>reserved</td>
<td>150 µs</td>
<td>reserved</td>
</tr>
<tr>
<td>4</td>
<td>reserved</td>
<td>reserved</td>
<td>reserved</td>
</tr>
<tr>
<td>5</td>
<td>reserved</td>
<td>reserved</td>
<td>reserved</td>
</tr>
<tr>
<td>6</td>
<td>6720 µs</td>
<td>reserved</td>
<td>reserved</td>
</tr>
<tr>
<td>7</td>
<td>reserved</td>
<td>250 µs</td>
<td>100 µs</td>
</tr>
</tbody>
</table>
1. CONFIGURATION

1.1. Hardware and network configuration

Your PC must be equipped with a Pentium 120 MHz and Windows 95. In this example we configure subscriber 25 to be connected to a WorldFIP network, operating at a bit rate of 1 Mbit/s with UTE formatted frames; the access to the network is through a FIELDREIVE component on a CC165 board equipped with a 40 MHz quartz. The board address inside the IO area is 0x280; the MICROFIP component absolute address in the host microprocessor addressing space is 0x0CA000 (address for a 32-bit architecture). The silence time–out value will be 150 μs and the turn–around time–out value will be 10 μs.

### Note

Don’t forget to reserve memory in config.sys.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPHYADR</td>
<td>0x0019</td>
<td>subscriber number</td>
</tr>
<tr>
<td>*volatile Adr_Base</td>
<td>0x0CA000</td>
<td>host addressing space</td>
</tr>
<tr>
<td>Option_for_Phyadr</td>
<td>0</td>
<td>subscriber number in KPHYADR</td>
</tr>
<tr>
<td>Turnaround_Time</td>
<td>0</td>
<td>10 μs</td>
</tr>
<tr>
<td>MAU_Type</td>
<td>0</td>
<td>FIELDREIVE MAU</td>
</tr>
<tr>
<td>Standard_type</td>
<td>0</td>
<td>UTE delimiters &amp; FCS</td>
</tr>
<tr>
<td>Speed</td>
<td>1</td>
<td>1 Mbit/s</td>
</tr>
<tr>
<td>T0</td>
<td>3</td>
<td>150 μs</td>
</tr>
<tr>
<td>Quartz</td>
<td>1</td>
<td>40 MHz</td>
</tr>
</tbody>
</table>
1.2. Communication variable configuration

var1, var3, var5, var7 will be declared consumable variables with sizes of 16 bytes each, for a promptness of 50 ms. var7 will be defined as a global variable associated with the global identifier 0x9802.

var0, var2, var4 will be declared produced variables with sizes of 16 bytes, and with a refreshment period of 250 ms.

var6 will be declared produced variable with a size of 8 bytes, and with an infinite refreshment period.

- Promptness_BankB : 0 promptness of 50 ms on var5
- Promptness_BankA : 0 promptness of 50 ms on var1, var3, var7
- Refreshment_BankB : 0 refreshment of 250 ms on var6
- Refreshment_BankA : 1 infinite refreshment on var0, var2, var4
- SEL_IDG : 1 var7 is global
- Global_ID : 0x9802 var7 identifier value
- Var_Sizes[8] : [16,16,16,16,16,16,8,16] size of variables

⇒ var0 will be produced when receiving an id_dat frame with the 0x0019 identifier
⇒ var2 will be produced when receiving an id_dat frame with the 0x0219 identifier
⇒ var4 will be produced when receiving an id_dat frame with the 0x0419 identifier
⇒ var6 will be produced when receiving an id_dat frame with the 0x0619 identifier

⇒ var1 will be consumed when receiving an id_dat frame with the 0x0119 identifier
⇒ var3 will be consumed when receiving an id_dat frame with the 0x0319 identifier
⇒ var5 will be consumed when receiving an id_dat frame with the 0x0519 identifier
⇒ var7 will be consumed when receiving an id_dat frame with the 0x9802 identifier
⇒ synchro variable will be detected when receiving an id_dat frame with the 0x9801 identifier

1.3. Messaging configuration

Messages will be produced when receiving an id_msg frame with identifier 0x0619. Messages will be consumed when receiving an rp_msg_ack or rp_msg_noack frame with identifiers from 0x0019 to 0x0F19. Messages will be received and transmitted with segment 0; and the transmitted messages will be repeated once if necessary.

- Number_Of_Retries : 1 message repeated once
- Num_Segment : 0 segment 0

1.4. Port configuration

No filter will be applied on the input port value.

- Input_Ports_filter : 0 no filter on input port

1.5. Interrupt configuration

All the possible interrupts will be authorized.

- ENABLE_IT : 0xFF all interrupts allowed

1.6. Synchronization identifier configuration

The synchronization identifier will be 0x9801.

- Synchro_ID : 0x9801 synchronization identifier value
1.7. Identification variable configuration

Profile : 0x01  
Classe  : 0x00  
Constructor_h : 0x00  
Constructor_l : 0x01  
Model_h : 0x01  
Model_l : 0x63  
Version : 0x01  
User : 0x12
2. APPLICATION

A very simple example is presented below to show you how to process a very small application using MICROFIP HANDLER services. The functions which can be called depend on the interrupt register. Reading of the interrupts is through the \texttt{mf\_read\_interrupt\_register} function. When a bit is set to 1 in the interrupt register the corresponding activity is processed.

In this example no control is requested, and the result of the function is not tested. The values of the different produced variables are the same. The messages are always sent to subscriber 0 on segment 0, and are requested with acknowledgement, with one retry if necessary. The ports are not used. After initialization, the produced variables and a message are written.
3. USER APPLICATION PROGRAM

```c
#include "conio.h"
#include "user_opt.h"
#include "mfhd.h"

#define ADR_HARD_COMPOSANT 0x0CA000
#define ADR_COMPOSANT (MEMOIRE_MF *) ADR_HARD_COMPOSANT

#define ACTIVE 0x01
#define IDLE 0x00

#define IRQSA_VA_STI_VAP_6 0x01
#define IRQSA_VA_STI_SYNCHRO 0x02
#define IRQSA_VA_STI_MST 0x04
#define IRQSA_VA_STI_MSR 0x08
#define IRQSA_VA_STI_VAR_1 0x10
#define IRQSA_VA_STI_VAR_3 0x20
#define IRQSA_VA_STI_VAR_5 0x40
#define IRQSA_VA_STI_VAR_7 0x80

#define cst_sz_msg 128
#define SEL_ACK 1

static unsigned char tab_dat_t[120]
unsigned char var_cpt_mpt;
unsigned char var_cpt_mst;
unsigned short i;

typedef struct {
    unsigned short Size;
    unsigned char  DATA_BUFFER[128];
} MF_MSG_RECEIVED;

MF_MSG_RECEIVED tab_dat_msg_r;

unsigned short size_msg;
static unsigned char tab_dat_msg_t[128];
```
MF_CONFIGURATION User_Configuration = {
    25  /* K_PHYADR */
    ADR_COMPOSANT,  /* Adr_Base */
    0,    /* Option_For_Phyadr */
    1    /* Number_Of_Retries */
    0,    /* MAU_Type */
    0,    /* Standard Type */
    1,    /* Speed */
    0,    /* Turnaround Time */
    3,    /* T0 */
    1,    /* SEL_IDG */
    0xFFFF  /* ENABLE_IT */
    0,    /* Promptness_BankA */
    0,    /* Promptness_BankB */
    1,    /* Refreshment_BankA */
    0,    /* Refreshment_BankB */
    0,    /* Input_Port_Filter */
    0x9802,    /* Global_ID */
    0x9801,    /* Synchro_ID */
    00,    /* Num_Segment */
    1,    /* Quartz */
    16,16,16,16,16,16,8,16 }; /* Var_Sizes */

MF_IDENTIFICATION User_Identification = {
    0x01,
    0x00,
    0x00,
    0x01,
    0x01,
    0x63,
    0x01,
    0x12};
void main() {

unsigned short cr16;
volatile unsigned char vec_irqsa;

/* CC165 initialization for PC 32 bits */
{
unsigned short IOBase = ADDR_HARD_COMPOSANT >> 12 ;
_outp (0x280 , IOBase) ;
}

/* MICROFIP initialization */
cr16 = mf_initialize_network (&User_Configuration,
&User_Identification);

/* transmitted buffer initialization */
for (i=0; i<16; i++) {
    varcpt_mpt++;
    tab_dat_t[i] = varcpt_mpt;
}

/* writing of var6 produced on ID 619h */
cr = mf_var_write_loc (6,tab_dat_t);

/* writing of var4 produced on ID 419h */
cr = mf_var_write_loc (4,tab_dat_t);

/* writing of var2 produced on ID 219h */
cr = mf_var_write_loc (2,tab_dat_t);

/* writing of var0 produced on ID 19h */
cr = mf_var_write_loc (0,tab_dat_t);

/* writing of a message */
size = cst_sz_msg;
*(tab_dat_msg_t) = 0; /* destl address */
*(tab_dat_msg_t + 1) = 0; /* destl address */
*(tab_dat_msg_t + 2) = 0; /* dest segment number */
*(tab_dat_msg_t + 3) = 0; /* srch address */
*(tab_dat_msg_t + 4) = 0x19; /* srcl address */
/* transmitted message initialization */
for (i=6; i<size_msg; i++) {
    *(tab_dat_msg_t + i) = (unsigned char)var_cpt_mst++;
}

/* writing of a message */
cr = mf_send_message (size, SEL_ACK, tab_dat_msg_t);

while (true) {

    vec_irqsa = mf_read_interrupt_register();
    if ((vec_irqsa) != 0) {

        /* if var1 has been received, the variable is read in the buffer tab_dat_r */
        if (((vec_irqsa & IRQSA_VA_STI_VAR_1) != IDLE) {
            cr = mf_var_read_loc(1, &tab_dat_r);
        }
        /* if var3 has been received, the variable is read in the buffer tab_dat_r */
        if (((vec_irqsa & IRQSA_VA_STI_VAR_3) != IDLE) {
            cr = mf_var_read_loc(3, &tab_dat_r);
        }
        /* if var5 has been received, the variable is read in the buffer tab_dat_r */
        if (((vec_irqsa & IRQSA_VA_STI_VAR_5) != IDLE) {
            cr = mf_var_read_loc(5, &tab_dat_r);
        }
        /* if var7 has been received, the variable is read in the buffer tab_dat_r */
        if (((vec_irqsa & IRQSA_VA_STI_VAR_7) != IDLE) {
            cr = mf_var_read_loc(7, &tab_dat_r);
        }

        /* if var6 has been produced, the produced variables are written with the data contained in the buffer tab_dat_t */
        if (((vec_irqsa & IRQSA_VA_STI_VAP) != IDLE) {
            for (i=0; i<16; i++) {
                var_cpt_mpt++;
                tab_dat_t[i] = var_cpt_mpt;
            }
            cr = mf_var_write_loc(6, tab_dat_t);
            cr = mf_var_write_loc(4, tab_dat_t);
            cr = mf_var_write_loc(2, tab_dat_t);
            cr = mf_var_write_loc(0, tab_dat_t);
        }
    }
}
/* if a message has been received, the value is read */
if ((vec_irqsa & IRQSA_VA_STI_MSR) != IDLE) {
  cr = mf_read_message(&tab_dat_msg_r);
}

/* if a message has been sent, the value of a new one is written from the buffer */
if ((vec_irqsa & IRQSA_VA_STI_MST) != IDLE) {
  size = cst_sz_msg;
  *(tab_dat_msg_t) = 0; /* desth address */
  *(tab_dat_msg_t + 1) = 0; /* destl address */
  *(tab_dat_msg_t + 2) = 0; /* dest segment number */
  *(tab_dat_msg_t + 3) = 0; /* srch address */
  *(tab_dat_msg_t + 4) = 0x19; /* srcl address */
  for (i=6;i<size_msg;i++) {
    *(tab_dat_msg_t + i) = (unsigned char)var_cpt_mst++;
  }
  cr = mf_send_message(size,SEL_ACK,tab_dat_msg_t);
}
4. CC165 INITIALIZATION ON A 16–BIT PC

#define ADR_HARD_COMPOSANT 0x0CA000000L
#define ADR_COMPOSANT (MEMOIRE_MF *) ADR_HARD_COMPOSANT

/* CC165 initialization for PC 16 bits*/
{
  unsigned
  long T;
  T=(ADR_HARD_COMPOSANT >>12)+(ADR_HARD_COMPOSANT & 0x0FFFF);
  unsigned short IOBase = (unsigned short) L >> 12);
  _outp (0x280 , IOBase);
}
Appendix C

Example of system configuration and application using the new functions

1. CONFIGURATION TO BE PROCESSED

1.1. Hardware and network configuration

A PC must be equipped with a Pentium 120 MHz and Windows 95. In this example, we configure subscriber 5 to be connected to a WorldFIP network, operating at a bit rate of 1 Mbit/s with UTE formatted frames; the access to the network is through a FIELDRIVE component on a CC165 board equipped with a 40 MHz quartz. The board address inside the IO area is 0x280; the MICROFIP component absolute address in the host microprocessor addressing space is 0x0CA000 (address for a 32–bit architecture). The silence time–out value will be 150 µs and the turn–around time–out value will be 10 µs.

Note

Don’t forget to reserve memory in config.sys.

<table>
<thead>
<tr>
<th>KPHYADR</th>
<th>0x05</th>
<th>subscriber number</th>
</tr>
</thead>
<tbody>
<tr>
<td>*volatile Adr_Base</td>
<td>0x0CA000</td>
<td>host addressing space</td>
</tr>
<tr>
<td>Option_for_Phyadr</td>
<td>0</td>
<td>subscriber number in KPHYADR</td>
</tr>
<tr>
<td>Turnaround_Time</td>
<td>0</td>
<td>10 µs</td>
</tr>
<tr>
<td>MAU_Type</td>
<td>0</td>
<td>FIELDRIVE MAU</td>
</tr>
<tr>
<td>Standard_type</td>
<td>0</td>
<td>UTE delimiters &amp; FCS</td>
</tr>
<tr>
<td>Startup_Channel</td>
<td>2</td>
<td>startup channel on presence and time–out</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 second</td>
</tr>
<tr>
<td>Msg_Rec_Mode</td>
<td>3</td>
<td>broadcasting and point–to–point reception</td>
</tr>
<tr>
<td>Speed</td>
<td>1</td>
<td>1 Mbit/s</td>
</tr>
<tr>
<td>T0</td>
<td>3</td>
<td>150 µs</td>
</tr>
<tr>
<td>Quartz</td>
<td>1</td>
<td>40 MHz</td>
</tr>
</tbody>
</table>
1.2. Communication variable configuration

var1, var3, var5, var7 will be declared consumable variables with sizes of 16, 16, 16, 8 bytes each, for a promptness of 250 ms.

var0, var2, var4, var6 will be declared produced variables with sizes of 16 bytes, and with a refreshment period of 250 ms.

- Promptness_BankB : 1 promptness of 250 ms on var5
- Promptness_BankA : 1 promptness of 250 ms on var1, var3, var7
- Refreshment_BankB : 0 refreshment of 250 ms on var6
- Refreshment_BankA : 1 infinite refreshment on var0, var2, var4
- SEL_IDG : 0 var7 is not global
- Global_ID : 0x0000 not used
- Synchrol_ID : 0x9801 synchro id value

→ var0 will be produced when receiving an id_dat frame with the 0x0005 identifier
→ var2 will be produced when receiving an id_dat frame with the 0x0205 identifier
→ var4 will be produced when receiving an id_dat frame with the 0x0405 identifier
→ var6 will be produced when receiving an id_dat frame with the 0x0605 identifier
→ var1 will be consumed when receiving an id_dat frame with the 0x0105 identifier
→ var3 will be consumed when receiving an id_dat frame with the 0x0305 identifier
→ var5 will be consumed when receiving an id_dat frame with the 0x0505 identifier
→ var7 will be consumed when receiving an id_dat frame with the 0x0705 identifier
→ synchro variable will be detected when receiving an id_dat frame with the 0x999 identifier

1.3. Messaging configuration

Messages will be produced when receiving an id_msg frame with identifier 0x0605. Messages will be consumed when receiving a rp_msg_ack or a rp_msg_noack frame with identifiers from 0x0005 to 0x0F05. Messages will be received and transmitted with segment number 0; and the transmitted messages will be repeated once if necessary.

- Number_Of_Retries : 1 message repeated once
- Num_Segment : 0 segment 0
1.4. Port configuration

No filter will be applied on the input port value.

\[
\text{Input\_Ports\_filter} : 3 \quad \text{filter on input port of 10 seconds}
\]

1.5. Interrupt configuration

All the possible interrupts will be authorized.

\[
\text{ENABLE\_IT} : 0xFF \quad \text{all interrupts authorized}
\]

1.6. Synchronization identifier configuration

The synchronization identifier will be 0x9801.

\[
\text{Synchro\_ID} : 0x9801 \quad \text{synchronization identifier value}
\]

1.7. Identification variable configuration

- Profile : 0x01
- Classe : 0x00
- Constructor_h : 0x00
- Constructor_l : 0x01
- Model_h : 0x01
- Model_l : 0x63
- Version : 0x01
- User : 0x13
2. APPLICATION

A very simple example is presented below to show you how to process a very small application using MICROFIP HANDLER services. The functions which can be called depend on the interrupt register. The interrupt reading is through the mf_read_interrupt_register function. When a bit is set to 1 in the interrupt register the corresponding activity is processed.

In this example no control is requested, and the result of the function is not tested. The values of the different produced variables are the same. The messages are always sent to subscriber 2 on segment 0, and are requested with acknowledgement, with one retry if necessary. The ports are not used.

After initialization, the produced variables and a message are written.

User application program

```c
#include <conio.h>
#include <string.h>
#include <stdio.h>
#include <process.h>
#include "user_opt.h"
#include "mfhd.h"

#define ADR_HARD_COMPOSANT 0x0CA000
#define ADR_COMPOSANT (MEMOIRE_MF *) ADR_HARD_COMPOSANT

#define IRQSA_VA_STI_VAP_6 0x01
#define IRQSA_VA_STI_SYNCHRO 0x02
#define IRQSA_VA_STI_MST 0x04
#define IRQSA_VA_STI_MSR 0x08
#define IRQSA_VA_STI_VAR_1 0x10
#define IRQSA_VA_STI_VAR_3 0x20
#define IRQSA_VA_STI_VAR_5 0x40
#define IRQSA_VA_STI_VAR_7 0x80

#define cst_sz_msg 128
#define cst_sz_mps 120
#define SEL_ACK 1

static unsigned char tab_dat_var_t[120];
unsigned char var_cpt_mpt;
unsigned char var_cpt_mst;
unsigned short i;

typedef struct {
  unsigned short Size;
  unsigned short DATA_BUFFER_[128];
} MF_MSG_RECEIVED;

MF_MSG_RECEIVED tab_dat_msg_r;
unsigned short size_msg;
static unsigned char tab_dat_msg_t[128];
```
Example of system configuration and application using the new functions

MF_NEW_CONFIGURATION User_New_Configuration = {
    5 /* K_PHYADR */
    ADR_COMPOSANT, /* ADR_Base */
    0, /* Option_For_Ph yadr */
    1 /* Number_Of_Retries */
    CONF_VA_SEL_COL_NO, /* MAU_Type */
    CONF_VA_SEL_CEI_NO, /* Standard Type */
    CONF_VA_STU_CHL_PRES_1s, /* valid voies pres top 1s */
    CONFA_VA_ENA_MSR_PAP_DIF /* msg recep diff & pt a pt */
    CONF_VA_SEL_FRO_1, /* Speed */
    CONF_VA_SEL_TRP_1, /* Turnaround Time = 10µs */
    CONF_C_VA_SEL_TOT_4, /* T0 = 150µs */
    0xFF, /* ENABLE_IT */
    MPSPR_VA_SEL_IFI_10, /* Input_Port_Filter 10seconds */
    00, /* Num_Segment */
    BASIC_VA_SEL_QTZ_40}; /* Quartz */

MF_VAR_CONF User_Var_Conf = {
    CONF_VA_SEL_IDG_NO, /* SEL_IDG */
    MPSPR_VA_SEL_APR_250, /* Promptness_BankA */
    MPSPR_VA_SEL_BPR_250, /* Promptness_BankB */
    MPSPR_VA_SEL_ARA_INF, /* Refreshment_BankA */
    MPSPR_VA_SEL_BRA_250, /* Refreshment_BankB */
    0x0000, /* Global_ID */
    0x9801, /* Synchro_ID */
    16,16,16,16,16,16,16,8 }; /* Var_Sizes */

MF_IDENTIFICATION User_Identification = {
    0x01,
    0x00,
    0x00,
    0x01,
    0x01,
    0x63,
    0x01,
    0x13};

void main() {
    unsigned char fin_test = 0;
    unsigned int i;
    volatile unsigned char vec_irqsa;

    /* INITIALISATION */

    unsigned short IOBase = ADR_HARD_COMPOSANT >> 12;
    _outp ( 0x280 , IOBase );
}

/* INITIALISATION REGISTRES */
cr16 = mf_new_initialize_network(&User_New_Configuration,
    &User_Identification);
printf("\tCR16_mf_new_initialization : %xH\n",cr16);
/* INITIALISATION VARIABLES *********************************************/

cr16 = mf_mps_var_conf(&User_Var_Conf);
printf("\tCR16_mf_mps_var_conf : %xH\n",cr16);

/* INITIALISATION BUFFER VARIABLES PRODUITES ***************************/

for (i=0; i<cst_sz_mps; i++) {
    if (var_cpt_mpt < 255) {
        var_cpt_mpt++;
    } else {
        var_cpt_mpt = 0;
    }
    tab_dat_var_t[i] = var_cpt_mpt;
}

/* ECRITURES VARIABLES PRODUITES *************************************/

cr = mf_var_write_loc ( 0,tab_dat_var_t );
for (i=0; i<size_msg; i++) {
    *(tab_dat_msg_t + i) = (unsigned char)var_cpt_mst++;
}

cr = mf_send_message(size_msg,CONFA_VA_SEL_ACK_YES,tab_dat_msg_t);

/************************** INIT APPLICATION PROGRAM **********************/

printf("\tABONNE : %xH\n",User_New_Configuration.K_PHYADR);

while ( fin_test == 0 ) {
    while (!_kbhit()) {
        printf("\tTAPPEZ UNE TOUCHE POUR TERMINER LE TEST \n\n");
        while (fin_test == 0) {
            while (!_kbhit()) {
                /* When PC interrupt is not used ******************************/
                /* interrupt register must be used in polling mode ***********/
            }
        }
    }
}
Example of system configuration and application using the new functions

```c
/* LECTURES REGISTRE D’INTERRUPTIONS *********************************/
vec_irqsa = mf_read_interrupt_register();
if ((vec_irqsa) != 0) {

/* IT SYNCHRO POSITIONNEE ? *******************************************/
if ((vec_irqsa & IRQSA_VA_STI_SYN) != IDLE) {
}

/* IT MESSAGE RECU POSITIONNEE ? *******************************************/
if ((vec_irqsa & IRQSA_VA_STI_MSR) != IDLE) {
    cr = mf_read_message(&tab_dat_msg_r);
}

/* IT MESSAGE EMIS POSITIONNEE ? *******************************************/
if ((vec_irqsa & IRQSA_VA_STI_MST) != IDLE) {
    size_msg = cst_sz_msg;
    *(tab_dat_msg_t) = 0; /* adresse desth */
    *(tab_dat_msg_t + 1) = 2; /* adresse destl */
    *(tab_dat_msg_t + 2) = 0; /* numero de segment */
    *(tab_dat_msg_t + 3) = 0; /* adresse srch */
    *(tab_dat_msg_t + 4) = 0x05; /* adresse srcl */
    for (i=6; i<size_msg; i++) {
        *(tab_dat_msg_t + i) = (unsigned char)var_cpt_mst++;
    }
    cr = mf_send_message(size_msg, CONFA_VA_SEL_ACK_YES, tab_dat_msg_t);
}

/* IT VARIABLE 6 PRODUITE POSITIONNEE ? ******************************/
if ((vec_irqsa & IRQSA_VA_STI_VAP) != IDLE) {
    number_of_its_var_prod++;
    for (i=0; i<120; i++) {
        var_cpt_mpt++;
        tab_dat_var_t[i] = var_cpt_mpt;
    }
    cr = mf_var_write_loc(6, tab_dat_var_t);
    cr = mf_var_write_loc(4, tab_dat_var_t);
    cr = mf_var_write_loc(2, tab_dat_var_t);
    cr = mf_var_write_loc(0, tab_dat_var_t);
}

/* IT VARIABLE 1 A CONSOMMER POSITIONNEE ? ******************************/
if ((vec_irqsa & IRQSA_VA_STI_VAR_1) != IDLE) {
    cr = mf_var_read_loc(1, &tab_dat_var_r);
}
```

Example of system configuration and application using the new functions

```c
/* IT VARIABLE 3 A CONSOMMER POSITIONNEE ? ***************************/
if ((vec_irqsa & IRQSA_VA_STI_VAR_3) != IDLE ) {
    cr = mf_var_read_loc(3,&tab_dat_var_r);
}

/* IT VARIABLE 5 A CONSOMMER POSITIONNEE ? ***************************/
if ((vec_irqsa & IRQSA_VA_STI_VAR_5) != IDLE ) {
    cr = mf_var_read_loc(5,&tab_dat_var_r);
}

/* IT VARIABLE 7 A CONSOMMER POSITIONNEE ? ***************************/
if ((vec_irqsa & IRQSA_VA_STI_VAR_7) != IDLE ) {
    cr = mf_var_read_loc(7,&tab_dat_var_r);
}

/* FIN D’UN TEST : AFFICHAGE COMPTEURS ***************************/
_getch();

printf("\tTAPPEZ q POUR TERMINER autre POUR CONTINUER LE TEST \n");
printf("\n");
while (!_kbhit()) {
    if ( _getch()=='q' ) {
        fin_test = 1; }
    else {
        printf("\tTAPPEZ UNE TOUCHE POUR TERMINER LE TEST \n");
        printf("\n");
    }
}

exit(0);
```

3. REDUNDANCY APPLICATION EXAMPLE

```c
#include <windows.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include "user_opt.h"
#include "mfhd.h"

#define IDLE 0x00

/* Messaging status */

#define IRQSA_VA_STI_VAP 0x01
#define IRQSA_VA_STI_SYN 0x02
#define IRQSA_VA_STI_MST 0x04
#define IRQSA_VA_STI_MSR 0x08
#define IRQSA_VA_STI_VAR_1 0x10
#define IRQSA_VA_STI_VAR_3 0x20
#define IRQSA_VA_STI_VAR_5 0x40
#define IRQSA_VA_STI_VAR_7 0x80
#define cst_sz_msg 128
#define cst_sz_mps 120

#define ADR_HARD_COMPOSANT 0x0ca000
#define ADR_COMPOSANT (MEMOIRE_MF *) ADR_HARD_COMPOSANT

static unsigned char  tab_dat_msg_t[128];
static unsigned char  tab_dat_var_t[120];

unsigned short  size_msg;
unsigned short  *var_size;

int   nb_err_hard = 0;
int   nb_var6_prod = 0;
int   nb_var1_cons = 0;
int   nb_var3_cons = 0;
int   nb_var5_cons = 0;
int   nb_var7_cons = 0;
int   nb_synchro = 0;
int   ch_tempo;

unsigned char   cr;
unsigned short  cr16;

unsigned char   var_cpt_mst;
unsigned char   var_cpt_mpt;

MF_VAR_DATA     tab_dat_var_r;
MF_MSG_RECEIVED tab_dat_msg_r;
```
Example of system configuration and application using the new functions

MF_CONFIGURATION User_Configuration = {
  0x5,       /* K_PHYADR */
  ADR_COMPOSANT,  /* Addr_Base */
  0,        /* Option_For_Phyadr */
  1,        /* Number_Of_Retries */
  CONFC_VA_SEL_COL_NO,  /* MAU_Type */
  CONFC_VA_SEL_CEI_NO,  /* Standard Type */
  CONFC_VA_SEL_FRQ_1,   /* Speed */
  CONFC_VA_SEL_TRP_1,   /* Turnaround Time */
  CONFD_VA_SEL_TOT_4,   /* T0 */
  CONFD_VA_SEL_IDG_YES, /* SEL_IDG */
  0xFF,      /* ENABLE_IT */
  MPSPR_VA_SEL_APR_250, /* Promptness_BankA */
  MPSPR_VA_SEL_BPR_250, /* Promptness_BankB */
  MPSPR_VA_SEL_ARA_INF, /* Refreshment_BankA */
  MPSPR_VA_SEL_BRA_250, /* Refreshment_BankB */
  MPSPR_VA_SELIFI_10,  /* Input_Port_Filter */
  0x254,     /* Global_ID */
  0x999,     /* Synchro_ID */
  00,        /* Num_Segment */
  BASIC_VA_SEL_QTZ_40, /* Quartz */
  16,16,16,16,16,16,16,8 }; /* Var_Sizes */

MF_NEW_CONFIGURATION User_New_Configuration = {
  0x10,       /* K_PHYADR */
  ADR_COMPOSANT,  /* Addr_Base */
  0,        /* Option_For_Phyadr */
  1,        /* Number_Of_Retries */
  CONFC_VA_SEL_COL_NO,  /* MAU_Type */
  CONFC_VA_SEL_CEI_NO,  /* Standard Type */
  CONFA_VA_STU_CHL_PRES_1S,  /* validation des voies pres et top 1s */
  CONFA_VA_ENA_MSR_PAP_DIF, /* mode de rec msg pt à pt et diff */
  CONFC_VA_SEL_FRQ_1,   /* Speed */
  CONFC_VA_SEL_TRP_1,   /* Turnaround Time */
  CONFD_VA_SEL_TOT_4,   /* T0 */
  0xFF,      /* ENABLE_IT */
  MPSPR_VA_SELIFI_10,  /* Input_Port_Filter */
  00,        /* Num_Segment */
  BASIC_VA_SEL_QTZ_40, /* Quartz */
  DEFAULT_T1,  /* T1 */
  DEFAULT_T2}; /* T2 */

MF_VAR_CONF User_Var_Conf = {
  CONFDA_VA_SEL_IDG_NO,  /* SEL_IDG */
  MPSPR_VA_SEL_APR_250, /* Promptness_BankA */
  MPSPR_VA_SEL_BPR_250, /* Promptness_BankB */
  MPSPR_VA_SEL_ARA_INF, /* Refreshment_BankA */
  MPSPR_VA_SEL_BRA_INF, /* Refreshment_BankB */
  0x0888,   /* Global_ID */
  0x09801,  /* Synchro_ID */
  16,16,16,16,16,16,16,8 }; /* Var_Sizes */
MF_IDENTIFICATION User_Identification = {
    0x01,
    0x00,
    0x00,
    0x01,
    0x01,
    0x63,
    0x01,
    0x13};

BOOL indic_demenote_arret_thread = FALSE;

/****************************************************************************
*          TACHE TIMER
****************************************************************************/

void Task_Timer (void)
{
    DWORD errorcode = NO_ERROR;
    unsigned char ch;

    for( ; indic_demenote_arret_thread == FALSE; ){
        Sleep(10);
        ch = mf_redundancy();
    }

    indic_demenote_arret_thread = FALSE;
    ExitThread(errorcode);
}

/****************************************************************************
*          DEBUT DU MAIN
****************************************************************************/

void main() {

    unsigned char fin_test = 0;
    unsigned int i;
    unsigned char vec_irqsa;

    typedef void(*Task_Process)(void);
    typedef struct{
        int nPriority;
        HANDLE hevent;
        DWORD IdThread;
        HANDLE hThread;
        Task_Process Routine;
    } MICROFIP_TASK_DATA;

    MICROFIP_TASK_DATA timertaskinfo;
void Task_Timer(void);

timertaskinfo.nPriority = THREAD_PRIORITY_NORMAL;
timertaskinfo.Routine = Task_Timer;

/******************************************************************************/
*    INITIALISATION
******************************************************************************/
{
    unsigned short IOBase = ADR_HARD_COMPOSANT >> 12;
    _outp ( 0x280 , IOBase );
}
printf("\t MICROFIP HANDLER RELEASE R1.6 06/12/2000

MICROFIP HANDLER RELEASE R1.6 06/12/2000\\n");
printf("\\n");
/* INITIALISATION REGISTRES ***************************************************/
#if ( WITH_NEW_MICROFIP == NO )
cr16 = mf_initialize_network(&User_Configuration,&User_Identification) ;
printf("\t CR16_mf_initialization : %xH\\n",cr16);
#endif

#if ( WITH_NEW_MICROFIP == YES )
cr16 = mf_new_initialize_network(&User_New_Configuration, &User_Identification);
printf("\t CR16_mf_new_initialization : %xH\\n",cr16);
printf("\t test gestion redondance <n") ;
#endif
/* INITIALISATION VARIABLES***************************************************/
cr16 = mf_mps_var_conf(&User_Var_Conf) ;
printf("\t CR16_mf_mps_var_conf : %xH\\n",cr16);
#endif

timertaskinfo.hThread = CreateThread( NULL, 0, (LPTHREAD_START_ROUTINE)Task_Timer, (LPVOID)NULL, CREATE_SUSPENDED, &timertaskinfo.IdThread));
if (timertaskinfo.hThread == NULL ) indic_demande_arret_thread = TRUE;
if (ResumeThread(timertaskinfo.hThread) == (DWORD) -1)
    indic_demande_arret_thread = TRUE;

/* INITIALISATION BUFFER VARIABLES PRODUITES ***********************************/
for (i=0; i<cst_sz_mps; i++) {
    if (var_cpt_mpt < 255) {
        var_cpt_mpt++;
    } else {
        var_cpt_mpt = 0;
    }
    tab_dat_var_t[i] = var_cpt_mpt;
}
/*ECRITURES VARIABLES PRODUITES *********************************************/

cr = mf_var_write_loc ( 0,tab_dat_var_t);
cr = mf_var_write_loc ( 2,tab_dat_var_t);
cr = mf_var_write_loc ( 4,tab_dat_var_t);
cr = mf_var_write_loc ( 6,tab_dat_var_t);

/*TEST ECRITURE D’UN MESSAGE pour 9000seg de 0810seg ***********************/

dimension_msg = cst_sz_msg;
*(tab_dat_msg_t) = 0x90; /* adresse desth */
*(tab_dat_msg_t + 1) = 0; /* adresse destl */
*(tab_dat_msg_t + 2) = 0; /* numero de segment */
*(tab_dat_msg_t + 3) = 8; /* adresse srch */
*(tab_dat_msg_t + 4) = 0x10; /* adresse srcl */
for (i=6;i<dimension_msg;i++) {
    *(tab_dat_msg_t + i) = (unsigned char)var_cpt_mst++;
}
cr = mf_send_message(dimension_msg,CONF_VA_SEL_ACK_NO,tab_dat_msg_t);

/*INITIALISATION ET ECRITURE D’UN MESSAGE********************************/

dimension_msg = cst_sz_msg;
*(tab_dat_msg_t) = 0; /* adresse desth */
*(tab_dat_msg_t + 1) = 2; /* adresse destl */
*(tab_dat_msg_t + 2) = 0; /* numero de segment */
*(tab_dat_msg_t + 3) = 0; /* adresse srch */
*(tab_dat_msg_t + 4) = 0x10; /* adresse srcl */
for (i=6;i<dimension_msg;i++) {
    *(tab_dat_msg_t + i) = (unsigned char)var_cpt_mst++;
}
cr = mf_send_message(dimension_msg,CONF_VA_SEL_ACK_YES,tab_dat_msg_t);

/******INIT APPLICATION PROGRAM *******************************************/

#if ( WITH_NEW_MICROFIP == YES )
printf("ABONNE : \%x\n",User_New_Configuration.K_PHYADR);
#else
printf("ABONNE : \%x\n",User_Configuration.K_PHYADR);
#endif

/*************************************************************************/
/* MASTER MODE ***********************************************************/
/***************************************************************************/

while ( fin_test == 0 ) {
    while (!_kbhit()) {
        /******** When PC interrupt is not used ***************************/
        /****** interrupt register must be used in polling mode ***********/
/* LECTURES REGISTRE D’INTERRUPTIONS ***********************************/

vec_irqsa = mf_read_interrupt_register();

/* test bit erreur dans MSGSA */
ch_tempo = mf_new_read_event();
if ((ch_tempo & 0x20) == 0x20) nb_err_hard++;

if ((vec_irqsa) != 0) {

    /* IT MESSAGE RECUS POSITIONNEE ? *******************/
    if ((vec_irqsa & IRQSA_VA_STI_MSR) != IDLE ) {
        cr = mf_read_message(&tab_dat_msg_r);
    }

    /* IT MESSAGE EMIS POSITIONNEE ? *******************/
    if ((vec_irqsa & IRQSA_VA_STI_MST) != IDLE ) {
        size_msg = cst_sz_msg;
        *(tab_dat_msg_t) = 0; /* adresse desth */
        *(tab_dat_msg_t + 1) = 2; /* adresse destl */
        *(tab_dat_msg_t + 2) = 0; /* numero de segment */
        *(tab_dat_msg_t + 3) = 0; /* adresse srch */
        *(tab_dat_msg_t + 4) = 0x10; /* adresse srcl */
        for (i=6;i<size_msg;i++)  {
            *(tab_dat_msg_t + i) = (unsigned char)var_cpt_mst++;
        }
        cr=mf_send_message(size_msg,CONFA_VA_SEL_ACK_YES,tab_dat_msg_t);
    }

    /* IT VARIABLE PRODUITE POSITIONNEE ? ***************/
    if ((vec_irqsa & IRQSA_VA_STI_VAP) != IDLE ) {
        for (i=0; i<120; i++) {
            var_cpt_mpt++;
            tab_dat_var_t[i] = var_cpt_mpt;
        }
        nb_var6_prod++;
        cr = mf_var_write_loc ( 6,tab_dat_var_t );
        //cr = mf_var_write_loc ( 4,tab_dat_var_t );
        //cr = mf_var_write_loc ( 2,tab_dat_var_t );
        //cr = mf_var_write_loc ( 0,tab_dat_var_t );
    }

    /* IT SYCHRO POSITIONNEE ? *****************************/
    if ((vec_irqsa & IRQSA_VA_STI_SYN) != IDLE ) {
        nb_synchro++;
    }
}
Example of system configuration and application using the new functions

/* IT VARIABLE 1 A CONSOMMER POSITIONNEE ? *******************/
if ((vec_irqsa & IRQSA_VA_STI_VAR_1) != IDLE ) {
    cr = mf_var_read_loc(1,&tab_dat_var_r);
    nb_var1_cons++;
}

/* IT VARIABLE 3 A CONSOMMER POSITIONNEE ? *******************/
if ((vec_irqsa & IRQSA_VA_STI_VAR_3) != IDLE ) {
    cr = mf_var_read_loc(3,&tab_dat_var_r);
    nb_var3_cons++;
}

/* IT VARIABLE 5 A CONSOMMER POSITIONNEE ? *******************/
if ((vec_irqsa & IRQSA_VA_STI_VAR_5) != IDLE ) {
    nb_var5_cons++;
    cr = mf_var_read_loc(5,&tab_dat_var_r);
}

/* IT VARIABLE 7 A CONSOMMER POSITIONNEE ? *******************/
if ((vec_irqsa & IRQSA_VA_STI_VAR_7) != IDLE ) {
    cr = mf_var_read_loc(7,&tab_dat_var_r);
    nb_var7_cons++;
}

/* FIN D’UN TEST : *******************************************************/
_getch();

// affichage compteurs
printf("\t nb_err_hard \%xH\n",nb_err_hard);
printf("\t nb_synchro \%xH\n",nb_synchro);
printf("\t nb_var6_prod \%xH\n",nb_var6_prod);
printf("\t nb_var1_cons \%xH\n",nb_var1_cons);
printf("\t nb_var3_cons \%xH\n",nb_var3_cons);
printf("\t nb_var5_cons \%xH\n",nb_var5_cons);
printf("\t nb_var7_cons \%xH\n",nb_var7_cons);
printf("\n");

printf("\t TAPPEZ UNE TOUCHE REPRENDRE LE TEST \n");
printf("\t TAPPEZ UNE TOUCHE POUR SORTIR DE L’APPLICATION \n");
printf("\n");

while (!_kbhit()) {
}

if ( _getch()==’q’ ) {
    fin_test = 1;
}
else {
    printf("\t TAPPEZ UNE TOUCHE POUR SORTIR DU TEST \n");
    printf("\n");
}

indic_demande_arret_thread = TRUE;
Sleep(2000);

if (!CloseHandle (timertaskinfo.hThread)){
    printf("\tERREUR CLOSE \n");
}else{
    printf("\tCLOSE OK \n");
}

exit(0);
}