



```

        #endif
        #if( defined( MOD_IO ) )
            teste_io,
        #endif
        #if( defined( MOD_ADC ) )
            teste_adc,
        #endif
        #if( defined( MOD_EEPROM_SPI ) )
            teste_eeprom_spi,
        #endif
        #if( defined( MOD_SRAM ) )
            teste_sram,
        #endif
        #if( defined( MOD_MOTOR ) )
            teste_motor,
        #endif
    };

    inicializa_fw();

    tx_string_flash_porta_serial( ( const unsigned char * )"TESTE DE FIRMWARE\r\n\r\n" ); i =
0;
while( 1 )
{
    ClrWdt();

    if( F_1MS )
    {
        F_1MS = 0;
    }

    if( i < total_elementos_array( exec ) )
    {
        if( !( *exec[ i ] )() )
        {
            i++;
        }
    }
    else
    {
        if( i != 0xFF )
        {
            #if( TESTE_FW == TESTE_FW_INFINITO )
                i = 0;
            #else
                i = 0xFF;
            #endif

            tx_string_flash_porta_serial( ( const unsigned char * )" \r\n\r\nFIM DO
TESTE DE FIRMWARE\r\n\r\n" );
            #endif
        }
    }
}
#endif

/*****\
* teste_uart
* Rotina de teste de UART: envia pela serial qualquer os dados que foram recebidos da
* mesma (loopback), porém com os caracteres em letra maiúscula
*
* Parâmetros: void
* Retorno : 1 teste em andamento, 0 teste finalizado
*****/ inline
unsigned char teste_uart( void )
{
    static unsigned char tempo;
    static enum
    {
        SM_TESTE_UART_INICIA = 0,
        SM_TESTE_UART_TESTA,
    } sm = SM_TESTE_UART_INICIA;

```

```

switch( sm )
{
case SM_TESTE_UART_INICIA:
    tx_string_flash_porta_serial( ( const unsigned char * ) "\r\n\r\nTeste UART\r\n" );
    tx_string_flash_porta_serial( ( const unsigned char * ) "-Loopback;\r\n" );
    tx_string_flash_porta_serial( ( const unsigned char * ) "-Tempo de teste: 5
segundos.\r\n\r\n" );

    tempo = 10;

    sm = SM_TESTE_UART_TESTA;
    break;

case SM_TESTE_UART_TESTA:
    if( F_PORTA_SERIAL_PCT_RX )
    {
        for( porta_serial.tam_tx = 0 ; porta_serial.tam_tx < porta_serial.tam_rx ;
porta_serial.tam_tx++ )
        {
            porta_serial.tx[ porta_serial.tam_tx ] = upper_chr( porta_serial.rx[
porta_serial.tam_tx ] );
        }
        tx_pacote_porta_serial();

        porta_serial.tam_rx = 0;

        F_PORTA_SERIAL_PCT_RX = 0;
    }

    if( F_1000MS )
    {
        F_1000MS = 0;

        tempo--;
        if( !tempo )
        {
            sm = SM_TESTE_UART_INICIA;

            return( 0 );
        }
    }
    break;
}

return( 1 );
}

/*****\
* teste_can *
* Rotina de teste do CAN: há 5 tipos de teste: *
* 1) TIPO_PONTA_CAN = PONTA_CAN_TX: o nó começa enviando a primeira mensagem 0x00001?0A*
* (? corresponde ao número do nó: ex. o nó 1 vai transmitir a mensagem 0x0000110A, o*
* nó 2 vai transmitir a mensagem 0x0000120A e assim por diante). *
* Após a transmissão da mensagem, inicia-se um temporizador. Se após 1 segundo este *
* nó não receber uma mensagem idêntica a que ele transmitiu um segundo atrás, então *
* ele transmite novamente a mesma mensagem. Caso o nó receba a mensagem idêntica a *
* que ele havia transmitido, então ele envia a próxima mensagem 0x00001?0B e assim *
* por diante. *
* A cada mensagem, é invertido o tipo de ID (STD ou EXT). Da mesma forma, a primeira*
* e a oitava mensagem transmitem um RTR (sem payload, apenas ID) ao invés de um fra-*
* me de dados. *
* 2) TIPO_PONTA_CAN = PONTA_CAN_LOOPBACK: o nó simplesmente fica aguardando a chegada*
* de uma mensagem CAN e a retransmite. Este modo em conjunto com o primeiro formam *
* um loopback. *
* 3) TIPO_PONTA_CAN = PONTA_CAN_TX_RX: o nó funciona semelhantemente ao PONTA_CAN_TX,*
* porém, a cada 1 segundo, envia a próxima mensagem, independente de ter recebido *
* uma mensagem idêntica a que ele havia transmitido no momento anterior. Este modo é*
* interessante de ser usado quando se liga vários nós todos configurados como PONTA_*
* CAN_TX. *
* 4) TIPO_PONTA_CAN = PONTA_CAN_TESTE_PACOTES: semelhante ao modo PONTA_CAN_TX_RX, po-*
* rém imprime apenas o contador de pacotes transmitidos e recebidos ao receber o co-*
* mando "info" pela serial. A taxa de transmissão também é mais rápida que no modo *

```

```

*   PONTA_CAN_TX_RX. Após TOTAL_SEGUNDOS_TX de transmissão, pára a transmissão. Como o*
*   contador_tx é de 32 bits, com uma transmissão a cada 20ms, daria para contar por *
*   mais de 3 dias. É enviado também na saída serial as informações dos registradores *
*   de contagem de erros de transmissão e de recepção. *
* 5) TIPO_PONTA_CAN = PONTA_CAN_SNIFFER: recebe todas as mensagens e imprime na serial *
*
* IMPORTANTE: *
* - Configuração de uma ponta como transmissor (TIPO_PONTA_CAN = PONTA_CAN_TX) e a ou- *
* tra ponta como loopback (TIPO_PONTA_CAN = PONTA_CAN_LOOPBACK). Pode-se também con- *
* figurar a ponta como transmissor e receptor ao mesmo tempo, onde a transmissão é *
* feita de tempos e tempos, tratando também a recepção (TIPO_PONTA_CAN = PONTA_CAN_ *
* TX_RX) *
* - Atentar-se à configuração dos filtros e máscaras de recepção, conforme os IDs que *
* serão transmitidos *
*
* Parâmetros: void *
* Retorno : 1 teste em andamento, 0 teste finalizado *
\******/ #if(
TIPO_SW == SW_TESTE )
Can can_app;
unsigned char FS_MSG_CAN_RX;

inline unsigned char teste_can( void )
{
    #if( defined( MOD_CAN ) )
        #define TOTAL_SEGUNDOS_TX( s )      ( ( s * 1000ul ) / 20ul )

        #define PONTA_CAN_TX                0
        #define PONTA_CAN_LOOPBACK          1
        #define PONTA_CAN_TX_RX             2
        #define PONTA_CAN_TESTE_PACOTES     3
        #define PONTA_CAN_SNIFFER           4
        #define TIPO_PONTA_CAN              PONTA_CAN_TX_RX

        #if( TIPO_PONTA_CAN != PONTA_CAN_SNIFFER )
            unsigned char buf;
        #endif
        #if( ( TIPO_PONTA_CAN != PONTA_CAN_LOOPBACK ) && \ (
            TIPO_PONTA_CAN != PONTA_CAN_SNIFFER ) )
            unsigned char tipo_msg;
            unsigned char bytes[ 8 ];

            static unsigned char tipo_id;
            static unsigned long id;
        #endif
        #if( TIPO_PONTA_CAN == PONTA_CAN_TESTE_PACOTES )
            static unsigned long contador_tx;
            static unsigned long contador_rx;
        #endif

        static unsigned char n;
        static union
        {
            struct
            {
                unsigned char tx : 1;
                unsigned char tx_info : 1;
                unsigned char : 6;
            } bits;

            unsigned char valor;
        } flags;
        static unsigned int tx_wait;
        static enum
        {
            SM_TESTE_CAN_INICIA = 0,
            SM_TESTE_CAN_TESTA,
            SM_TESTE_CAN_OCIOSO,
        } sm = SM_TESTE_CAN_INICIA;

        switch( sm )
        {
            case SM_TESTE_CAN_INICIA:
                #if( TIPO_PONTA_CAN == PONTA_CAN_TX )

```

```

        tx_string_flash_porta_serial( ( const unsigned char * )"\r\n\r\nTeste CAN
- TX\r\n" );
        #elif( TIPO_PONTA_CAN == PONTA_CAN_LOOPBACK )
        tx_string_flash_porta_serial( ( const unsigned char * )"\r\n\r\nTeste CAN
- LOOPBACK\r\n" );
        #elif( TIPO_PONTA_CAN == PONTA_CAN_TX_RX )
        tx_string_flash_porta_serial( ( const unsigned char * )"\r\n\r\nTeste CAN
- TX/RX\r\n" );
        #elif( TIPO_PONTA_CAN == PONTA_CAN_TESTE_PACOTES )
        tx_string_flash_porta_serial( ( const unsigned char * )"\r\n\r\nTeste CAN
- contador de pacotes TX e RX\r\n" );
        tx_string_flash_porta_serial( ( const unsigned char * )"-Para obter os
contadores TX e RX, envie o comando \"info\"\r\n" );
        #elif( TIPO_PONTA_CAN == PONTA_CAN_SNIFFER )
        tx_string_flash_porta_serial( ( const unsigned char * )"\r\n\r\nTeste CAN
- Sniffer\r\n" );
        #endif
        tx_string_flash_porta_serial( ( const unsigned char * )"-Para sair do teste,
envie o comando \"fim\".\r\n\r\n" );

        n = 0;
        flags.valor = 0x00;
        tx_wait = 0;

        #if( TIPO_PONTA_CAN == PONTA_CAN_TESTE_PACOTES )
            contador_tx = 0;
            contador_rx = 0;
        #endif

        porta_serial.tam_rx = 0;
        F_PORTA_SERIAL_PCT_RX = 0;

        FS_MSG_CAN_RX = 0;

// Entra em modo de configuração e configura uma máscara e um filtro para o
teste
modo_configuracao_can();
configura_mascara_rx_can( 2, 0x00000000, CAN_MIDE_ID_EXATO, CAN_EXIDE_ID_EXT );
;
configura_filtro_rx_can( 14, 0x00000000, CAN_EXIDE_ID_EXT, CAN_BUFFER_RX_FIFO,
2 );
modo_operacao_can();

// Define o gerenciador de tratamento das mensagens CAN recebidas
configura_func_rx_can( teste_can_trata_msg_can_rx );

sm = SM_TESTE_CAN_TESTA;
break;

case SM_TESTE_CAN_TESTA:
    #if( ( TIPO_PONTA_CAN == PONTA_CAN_TX_RX ) || \
        ( TIPO_PONTA_CAN == PONTA_CAN_TESTE_PACOTES ) || \
        ( TIPO_PONTA_CAN == PONTA_CAN_SNIFFER ) )
        if( FS_MSG_CAN_RX )
        {
            FS_MSG_CAN_RX = 0;

            #if( TIPO_PONTA_CAN == PONTA_CAN_TESTE_PACOTES )
                contador_rx++;
            #else
                teste_can_imprime_info_can( 0, can_app.info_msg.num, can_app.
info_msg.id, can_app.info_msg.tipo_id, can_app.info_msg.tipo_msg, can_app.info_msg.
bytes, can_app.info_msg.qtd_bytes );
            #endif
        }

        #if( ( TIPO_PONTA_CAN == PONTA_CAN_TX_RX ) || \
            ( TIPO_PONTA_CAN == PONTA_CAN_TESTE_PACOTES ) )
            if( F_1MS )
            {
                F_1MS = 0;

                if( flags.bits.tx )
                {

```

```

        flags.bits.tx = 0;

        teste_can_carrega_msg_can_tx( n, &buf, &id, &tipo_id, &
tipo_msg, bytes );

        buf = 0;

        #if( TIPO_PONTA_CAN == PONTA_CAN_TESTE_PACOTES )
            contador_tx++;
        #else
            teste_can_imprime_info_can( 1, buf, id, tipo_id, tipo_msg,
bytes, 8 );
        #endif

        prepara_id_tx_can( buf, id, tipo_id, tipo_msg );
        prepara_dados_tx_can( buf, bytes, 8 );
        tx_can( buf, CAN_PRIORIDADE_BAISSIMA + ( rand() % 4 ) );

        #if( TIPO_PONTA_CAN == PONTA_CAN_TESTE_PACOTES )
            tx_wait = 5;
        #else
            tx_wait = 1000;
        #endif
    }
    else
    {
        if( tx_wait )
        {
            tx_wait--;
        }

        if( !tx_wait )
        {
            #if( TIPO_PONTA_CAN == PONTA_CAN_TESTE_PACOTES )
                if( contador_tx < TOTAL_SEGUNDOS_TX( 3600 ) )
                {
                    flags.bits.tx = 1;
                }
            #else
                {
                    if( !F_PORTA_SERIAL_TX )
                    {
                        flags.bits.tx_info = 0;

                        insere_string_flash_porta_serial( ( const
unsigned char * )"fim\r\n" );

                        tx_pacote_porta_serial();
                    }

                    sm = SM_TESTE_CAN_OCIOSO;
                }
            #else
                flags.bits.tx = 1;
            #endif

            if( ++n == 10 )
            {
                n = 0;
            }
        }
    }
}
#endif
#else
    #if( TIPO_PONTA_CAN == PONTA_CAN_TX )
        if( FS_MSG_CAN_RX )
        {
            FS_MSG_CAN_RX = 0;

            teste_can_imprime_info_can( 0, can_app.info_msg.num, can_app.
info_msg.id, can_app.info_msg.tipo_id, can_app.info_msg.tipo_msg, can_app.info_msg.
bytes, can_app.info_msg.qtd_bytes );

            if( tipo_id == CAN_ID_STD )
            {

```

```

        id &= 0x000007FF;
    }

    if( can_app.info_msg.id == id )
    {
        if( ++n == 10 )
        {
            n = 0;
        }

        tx_wait = 1000;
    }
}

if( F_LMS )
{
    F_LMS = 0;

    if( flags.bits.tx )
    {
        flags.bits.tx = 0;

        teste_can_carrega_msg_can_tx( n, &buf, &id, &tipo_id, &
tipo_msg, bytes );

        buf = 0;
        teste_can_imprime_info_can( 1, buf, id, tipo_id, tipo_msg,
bytes, 8 );

        prepara_id_tx_can( buf, id, tipo_id, tipo_msg );
        prepara_dados_tx_can( buf, bytes, 8 );
        tx_can( buf, CAN_PRIORIDADE_BAISSIMA + ( rand() % 4 ) );

        tx_wait = 1000;
    }
    else
    {
        if( tx_wait )
        {
            tx_wait--;
        }

        if( !tx_wait )
        {
            flags.bits.tx = 1;
        }
    }
}
#else
if( FS_MSG_CAN_RX )
{
    FS_MSG_CAN_RX = 0;

    teste_can_imprime_info_can( 0, can_app.info_msg.num, can_app.
info_msg.id, can_app.info_msg.tipo_id, can_app.info_msg.tipo_msg, can_app.info_msg.
bytes, can_app.info_msg.qtd_bytes );

    // Aponta para o buffer de transmissão adequado, dos que foram
configurados para tal tarefa (0 e 1)
    buf = n & 0x01;
    buf = 0;

    if( ++n == 10 )
    {
        n = 0;
    }

    teste_can_imprime_info_can( 1, buf, can_app.info_msg.id, can_app.
info_msg.tipo_id, can_app.info_msg.tipo_msg, can_app.info_msg.bytes, can_app.info_msg.
qtd_bytes );

    prepara_id_tx_can( buf, can_app.info_msg.id, can_app.info_msg.
tipo_id, can_app.info_msg.tipo_msg );
    prepara_dados_tx_can( buf, can_app.info_msg.bytes, can_app.

```

```

info_msg.qtd_bytes );
        tx_can( buf, CAN_PRIORIDADE_BAIXISSIMA + ( rand() % 4 ) );
    }
    #endif
#endif
break;

case SM_TESTE_CAN_OCIOSO:
    break;
}

if( sm > SM_TESTE_CAN_INICIA )
{
    if( F_PORTA_SERIAL_PCT_RX )
    {
        F_PORTA_SERIAL_PCT_RX = 0;

        if( pos_str_flash_buffer( ( const unsigned char * )"fim", porta_serial.rx,
porta_serial.tam_rx ) != -1 )
        {
            porta_serial.tam_rx = 0;

            sm = SM_TESTE_CAN_INICIA;

            return( 0 );
        }

        #if( TIPO_PONTA_CAN == PONTA_CAN_TESTE_PACOTES )
            if( pos_str_flash_buffer( ( const unsigned char * )"info",
porta_serial.rx, porta_serial.tam_rx ) != -1 )
            {
                flags.bits.tx_info = 1;
            }
        #endif

        porta_serial.tam_rx = 0;
    }

    #if( TIPO_PONTA_CAN == PONTA_CAN_TESTE_PACOTES )
        if( flags.bits.tx_info )
        {
            if( !F_PORTA_SERIAL_TX )
            {
                flags.bits.tx_info = 0;

                insere_string_flash_porta_serial( ( const unsigned char * )
"contador_tx = " );
                dec_para_ascii( contador_tx, buffer_ascii );
                insere_byte_porta_serial( buffer_ascii[ 0 ] );
                insere_byte_porta_serial( buffer_ascii[ 1 ] );
                insere_byte_porta_serial( buffer_ascii[ 2 ] );
                insere_byte_porta_serial( buffer_ascii[ 3 ] );
                insere_byte_porta_serial( buffer_ascii[ 4 ] );
                insere_byte_porta_serial( buffer_ascii[ 5 ] );
                insere_byte_porta_serial( buffer_ascii[ 6 ] );
                insere_byte_porta_serial( buffer_ascii[ 7 ] );
                insere_string_flash_porta_serial( ( const unsigned char * )"r\n"
);
                insere_string_flash_porta_serial( ( const unsigned char * )
"contador_rx = " );
                dec_para_ascii( contador_rx, buffer_ascii );
                insere_byte_porta_serial( buffer_ascii[ 0 ] );
                insere_byte_porta_serial( buffer_ascii[ 1 ] );
                insere_byte_porta_serial( buffer_ascii[ 2 ] );
                insere_byte_porta_serial( buffer_ascii[ 3 ] );
                insere_byte_porta_serial( buffer_ascii[ 4 ] );
                insere_byte_porta_serial( buffer_ascii[ 5 ] );
                insere_byte_porta_serial( buffer_ascii[ 6 ] );
                insere_byte_porta_serial( buffer_ascii[ 7 ] );
                insere_string_flash_porta_serial( ( const unsigned char * )"r\n"
);
            }

            insere_string_flash_porta_serial( ( const unsigned char * )
"erros_tx = " );
            dec_para_ascii( C1ECbits.TERRCNT, buffer_ascii );

```



```

        insere_byte_porta_serial( buffer_ascii[ 0 ] );
        insere_byte_porta_serial( buffer_ascii[ 1 ] );
        insere_byte_porta_serial( buffer_ascii[ 2 ] );
        insere_byte_porta_serial( buffer_ascii[ 3 ] );
        insere_byte_porta_serial( buffer_ascii[ 4 ] );
        insere_byte_porta_serial( buffer_ascii[ 5 ] );
        insere_byte_porta_serial( buffer_ascii[ 6 ] );
        insere_byte_porta_serial( buffer_ascii[ 7 ] );
        insere_byte_porta_serial( ' ' ); dec_para_ascii(
        ClTERRCNT, buffer_ascii );
        insere_byte_porta_serial( buffer_ascii[ 0 ] );
        insere_byte_porta_serial( buffer_ascii[ 1 ] );
        insere_byte_porta_serial( buffer_ascii[ 2 ] );
        insere_byte_porta_serial( buffer_ascii[ 3 ] );
        insere_byte_porta_serial( buffer_ascii[ 4 ] );
        insere_byte_porta_serial( buffer_ascii[ 5 ] );
        insere_byte_porta_serial( buffer_ascii[ 6 ] );
        insere_byte_porta_serial( buffer_ascii[ 7 ] );
        insere_string_flash_porta_serial( ( const unsigned char * ) "\r\n"
);
        insere_string_flash_porta_serial( ( const unsigned char * )
"erros_rx = " );
        dec_para_ascii( ClECbits.RERRCNT, buffer_ascii );
        insere_byte_porta_serial( buffer_ascii[ 0 ] );
        insere_byte_porta_serial( buffer_ascii[ 1 ] );
        insere_byte_porta_serial( buffer_ascii[ 2 ] );
        insere_byte_porta_serial( buffer_ascii[ 3 ] );
        insere_byte_porta_serial( buffer_ascii[ 4 ] );
        insere_byte_porta_serial( buffer_ascii[ 5 ] );
        insere_byte_porta_serial( buffer_ascii[ 6 ] );
        insere_byte_porta_serial( buffer_ascii[ 7 ] );
        insere_byte_porta_serial( ' ' );
        dec_para_ascii( ClRERRCNT, buffer_ascii ); insere_byte_porta_serial(
        buffer_ascii[ 0 ] ); insere_byte_porta_serial( buffer_ascii[ 1 ] );
        insere_byte_porta_serial( buffer_ascii[ 2 ] );
        insere_byte_porta_serial( buffer_ascii[ 3 ] );
        insere_byte_porta_serial( buffer_ascii[ 4 ] );
        insere_byte_porta_serial( buffer_ascii[ 5 ] );
        insere_byte_porta_serial( buffer_ascii[ 6 ] );
        insere_byte_porta_serial( buffer_ascii[ 7 ] );
        insere_string_flash_porta_serial( ( const unsigned char * ) "\r\n"
);
        tx_pacote_porta_serial();
    }
    }
    #endif
}

return( 1 );
#endif

return( 0 );
}

/*****\
* teste_can_trata_msg_can_rx                               *
* Rotina de tratamento de mensagens CAN recebidas         *
*                                                         *
* Parâmetros: void                                       *
* Retorno : void                                         *
*****/ inline
void teste_can_trata_msg_can_rx( void )
{
    can_app = can;

    if( can.info_int.flags.bits.ovf )
    {
        can.info_int.flags.bits.ovf = 0;

        tx_string_flash_porta_serial( ( const unsigned char * ) "OVERFLOW!\r\n");
    }
    else

```

```

    {
        F_CAN_MSG_RX = 0;
        FS_MSG_CAN_RX = 1;
    }
}

/*****\
* teste_can_carrega_msg_can_tx *
* Rotina de geração de informações da mensagem CAN que será transmitida *
* *
* Parâmetros: número sequencial que define o payload da mensagem, bem como ID, tipo de *
* ID e tipo de mensagem *
* ponteiro para o buffer que será escolhido para realizar a transmissão *
* ponteiro para o ID que será gerado para a mensagem *
* ponteiro para o tipo de ID que será gerado para a mensagem *
* ponteiro para o tipo de mensagem que será gerado para a mensagem *
* ponteiro para o payload que será gerado para a mensagem *
* Retorno : void *
\*****/ void
teste_can_carrega_msg_can_tx( unsigned char seq, unsigned char *buf_tx, unsigned long
    *id, unsigned char *tipo_id, unsigned char *tipo_msg, unsigned char *payload )
{
    unsigned char i;

    // Aponta para o buffer de transmissão adequado, dos que foram configurados para tal
    tarefa (0 e 1)
    *buf_tx = seq & 0x01;

    // Carrega o payload
    for( i = 0 ; i < 8 ; i++ )
    {
        payload[ i ] = seq;
    }

    // id<9..8>: número no nó
    switch( seq )
    {
    case 0:
        *id      = 0x0000110A;
        *tipo_id = CAN_ID_STD;
        *tipo_msg = CAN_MSG_RTR;
        break;

    case 1:
        *id      = 0x0000110B;
        *tipo_id = CAN_ID_EXT;
        *tipo_msg = CAN_MSG_DADOS;
        break;

    case 2:
        *id      = 0x0000110C;
        *tipo_id = CAN_ID_STD;
        *tipo_msg = CAN_MSG_DADOS;
        break;

    case 3:
        *id      = 0x0000110D;
        *tipo_id = CAN_ID_EXT;
        *tipo_msg = CAN_MSG_DADOS;
        break;

    case 4:
        *id      = 0x0000110E;
        *tipo_id = CAN_ID_STD;
        *tipo_msg = CAN_MSG_DADOS;
        break;

    case 5:
        *id      = 0x0000110F;
        *tipo_id = CAN_ID_EXT;
        *tipo_msg = CAN_MSG_DADOS;
        break;
    }
}

```

```

    case 6:
        *id      = 0x00001110;
        *tipo_id  = CAN_ID_STD;
        *tipo_msg = CAN_MSG_DADOS;
        break;

    case 7:
        *id      = 0x00001111;
        *tipo_id  = CAN_ID_EXT;
        *tipo_msg = CAN_MSG_RTR;
        break;

    case 8:
        *id      = 0x00001112;
        *tipo_id  = CAN_ID_STD;
        *tipo_msg = CAN_MSG_DADOS;
        break;

    case 9:
        *id      = 0x00001113;
        *tipo_id  = CAN_ID_EXT;
        *tipo_msg = CAN_MSG_DADOS;
        break;
}

/*****\
* teste_can_imprime_info_can *
* Rotina de impressão dos dados de mensagens CAN enviadas e recebidas *
* *
* Parâmetros: mensagem transmitida (tx=1) ou recebida (tx=0), número do buffer, ID, tipo *
* de ID (CAN_ID_STD ou CAN_ID_EXT), tipo de mensagem (CAN_MSG_RTR ou CAN_ *
* MSG_DADOS), payload e tamanho do payload *
* Retorno : void *
\*****/ void
teste_can_imprime_info_can( unsigned char tx, unsigned char num_buf, unsigned long id
, unsigned char tipo_id, unsigned char tipo_msg, unsigned char *bytes, unsigned char
qtd_bytes )
{
    unsigned char i;
    unsigned char hex[ 2 ];
    union unsigned_char aux;

    if( tx )
    {
        tx_string_flash_porta_serial( ( const unsigned char * )"Msg CAN TX - buffer " );
    }
    else
    {
        tx_string_flash_porta_serial( ( const unsigned char * )"Msg CAN RX - buffer " );
    }
    tx_byte_porta_serial( '0' + ( num_buf / 10 ) );
    tx_byte_porta_serial( '0' + ( num_buf % 10 ) );

    if( tipo_msg == CAN_MSG_RTR )
    {
        tx_string_flash_porta_serial( ( const unsigned char * )" (RTR ) [ID " );
        if( tipo_id == CAN_ID_STD )
        {
            tx_string_flash_porta_serial( ( const unsigned char * )"STD" );
        }
        else
        {
            tx_string_flash_porta_serial( ( const unsigned char * )"EXT" );
        }
        tx_string_flash_porta_serial( ( const unsigned char * )" = " );
    }

    for( i = 0 ; i < 4 ; i++ )
    {
        aux.value = id >> ( 8 * ( 3 - i ) );

        byte_para_hex( aux.value, hex );
        tx_byte_porta_serial( hex[ 0 ] );
    }
}

```

```

        tx_byte_porta_serial( hex[ 1 ] );
    }
    tx_string_flash_porta_serial( ( const unsigned char * )" ] " );

    for( i = 0 ; i < 8 ; i++ )
    {
        tx_byte_porta_serial( ' ' );
        tx_byte_porta_serial( ' ' );
        tx_byte_porta_serial( ' ' );
    }
}
else
{
    tx_string_flash_porta_serial( ( const unsigned char * )" (DADOS) [ID " );
    if( tipo_id == CAN_ID_STD )
    {
        tx_string_flash_porta_serial( ( const unsigned char * )"STD" );
    }
    else
    {
        tx_string_flash_porta_serial( ( const unsigned char * )"EXT" );
    }
    tx_string_flash_porta_serial( ( const unsigned char * )" = " );

    for( i = 0 ; i < 4 ; i++ )
    {
        aux.value = id >> ( 8 * ( 3 - i ) );

        byte_para_hex( aux.value, hex );
        tx_byte_porta_serial( hex[ 0 ] );
        tx_byte_porta_serial( hex[ 1 ] );
    }
    tx_string_flash_porta_serial( ( const unsigned char * )" [ " );

    for( i = 0 ; i < 8 ; i++ )
    {
        if( i < qtd_bytes )
        {
            aux.value = bytes[ i ];

            byte_para_hex( aux.value, hex );
        }
        else
        {
            hex[ 0 ] = ' ';
            hex[ 1 ] = ' ';
        }

        tx_byte_porta_serial( hex[ 0 ] );
        tx_byte_porta_serial( hex[ 1 ] );

        if( i < 7 )
        {
            tx_byte_porta_serial( ' ' );
        }
    }
    tx_string_flash_porta_serial( ( const unsigned char * )" ] " );
}

if( !tx )
{
    tx_string_flash_porta_serial( ( const unsigned char * )" ICODE: " );
    for( i = 0 ; i < 8 ; i++ )
    {
        tx_byte_porta_serial( ( !( can.info_int.icode & ( 0x01 << ( 7 - i ) ) ) ) ? '0'
: '1' );
    }

    tx_string_flash_porta_serial( ( const unsigned char * )" , err: " );
    tx_byte_porta_serial( '0' + can.info_int.flags.bits.err );

    tx_string_flash_porta_serial( ( const unsigned char * )" , ovf: " );
    tx_byte_porta_serial( '0' + can.info_int.flags.bits.ovf );
}

```



```

    *ptr++ = 1;          // id_orig[2]

    *ptr++ = 0;          // id_dest[0]
    *ptr++ = 0;          // id_dest[1]
    *ptr++ = 0;          // id_dest[2]

    *ptr++ = 'P';        // dados[0]
    *ptr++ = 'A';        // dados[1]
    *ptr++ = 'C';        // dados[2]
    *ptr++ = 'O';        // dados[3]
    *ptr++ = 'T';        // dados[4]
    *ptr++ = 'E';        // dados[5]
    *ptr++ = ' ';        // dados[6]
    *ptr++ = 'T';        // dados[7]
    *ptr++ = 'X';        // dados[8]
    *ptr++ = '/';        // dados[9]
    *ptr++ = 'R';        // dados[10]
    *ptr++ = 'X';        // dados[11]

    *ptr++ = 0;          // CKS, será calculado no prepara_payload_rf()
#endif

porta_serial.tam_rx = 0;
F_PORTA_SERIAL_PCT_RX = 0;

#if( ( MODO_TESTE_RF == TESTE_RF_SNIFFER ) || \
      ( MODO_TESTE_RF == TESTE_RF_LOOPBACK ) || \
      ( MODO_TESTE_RF == TESTE_RF_TX ) )
    #if( MODO_TESTE_RF == TESTE_RF_SNIFFER )
        configura_preambulo_rf( 0, 0x00 );
    #endif
    configura_modulo_rf( RF_MODALO_RX );
#endif

sm = SM_TESTE_RF_TESTA;
break;

case SM_TESTE_RF_TESTA:
    #if( MODO_TESTE_RF == TESTE_RF_SNIFFER )
        if( novo_payload_rf() )
        {
            recebe_payload_rf();

            tx_byte_porta_serial( '#' );
            tx_byte_porta_serial( '#' );
            for( i = 0 ; i < rf.rx.tamanho ; i++ )
                tx_byte_porta_serial( rf.rx.payload[ i ] );
            tx_byte_porta_serial( '\r' );
            tx_byte_porta_serial( '\n' );
        }

        if( F_PORTA_SERIAL_PCT_RX )
        {
            F_PORTA_SERIAL_PCT_RX = 0;

            if( pos_str_flash_buffer( ( const unsigned char * )"fim", porta_serial
.rx, porta_serial.tam_rx ) != -1 )
            {
                porta_serial.tam_rx = 0;

                sm = SM_TESTE_CAN_INICIA;

                return( 0 );
            }

            porta_serial.tam_rx = 0;
        }
    #elif( MODO_TESTE_RF == TESTE_RF_LOOPBACK )
        if( novo_payload_rf() )
        {
            recebe_payload_rf();

            tx_byte_porta_serial( '#' );
            tx_byte_porta_serial( '#' );

```

```

        for( i = 0 ; i < rf.rx.tamanho ; i++ )
            tx_byte_porta_serial( rf.rx.payload[ i ] );
        tx_byte_porta_serial( '\r' );
        tx_byte_porta_serial( '\n' );

        // Retransmite o pacote recebido
        copia_bytes( ( unsigned char * )&rf.rx, ( unsigned char * )&rf.tx,
sizeof( rf.tx ) );

        prepara_payload_rf();
        tx_rf( ID_SENSOR );
    }

    if( F_1000MS )
    {
        F_1000MS = 0;

        tempo--;
        if( !tempo )
        {
            sm = SM_TESTE_RF_INICIA;

            return( 0 );
        }
    }
    #elif ( MODO_TESTE_RF == TESTE_RF_TX )

    if( F_500MS )
    {
        F_500MS = 0;

        prepara_payload_rf();
        tx_rf( ID_SENSOR );
    }

    if( novo_payload_rf() )
    {
        recebe_payload_rf();

        tx_byte_porta_serial( '#' );
        tx_byte_porta_serial( '#' );
        for( i = 0 ; i < rf.rx.tamanho ; i++ )
            tx_byte_porta_serial( rf.rx.payload[ i ] );
        tx_byte_porta_serial( '\r' );
        tx_byte_porta_serial( '\n' );
    }

    if( F_1000MS )
    {
        F_1000MS = 0;

        tempo--;
        if( !tempo )
        {
            sm = SM_TESTE_RF_INICIA;

            return( 0 );
        }
    }
    #endif
    break;
}

return( 1 );
#endif

return( 0 );
}
#endif

/*****\
* teste_eeprom_spi
* Escreve todos os bytes da EEPROM da memória com o valores de 0 a 255. Ao final da es-*
```

```

* crita, lê byte a byte e compara se o valor lido é igual ao esperado. Caso seja dife- *
* rente, mostra endereço, valor esperado e valor lido *
* *
* Parâmetros: void *
* Retorno : 1 teste em andamento, 0 teste finalizado *
\*****/ #if(
TIPO_SW == SW_TESTE )
inline unsigned char teste_eeeprom_spi( void )
{
    #if( defined( MOD_EEPROM_SPI ) )
        #define TOTAL_BANCOS_MEMORIA 1

        volatile unsigned long i, j, end;
        unsigned char val;
        unsigned char esp;
        unsigned char ok;
        unsigned char igual;
        unsigned int rnd;
        static unsigned int end_rnd[ 100 ];
        static enum
        {
            SM_TESTE_E2P_INICIA = 0,
            SM_TESTE_E2P_TESTA,
        } sm = SM_TESTE_E2P_INICIA;

        switch( sm )
        {
        case SM_TESTE_E2P_INICIA:
            tx_string_flash_porta_serial( ( const unsigned char * )"\r\n\r\nTeste de E2P:\r\n" );

            srand( obtem_tick() );
            for( i = 0 ; i < 100 ; i++ )
            {
                ok = 1;
                do
                {
                    ClrWdt();

                    rnd = ( unsigned int )rand();

                    igual = 1;
                    for( j = 0 ; j < i ; j++ )
                    {
                        if( end_rnd[ j ] == rnd )
                        {
                            ok = 0;
                            igual = 0;
                            j = i;
                        }
                    }

                    if( igual )
                    {
                        ok = 1;
                    }
                }
                while( !ok );

                end_rnd[ i ] = rnd;
            }

            sm = SM_TESTE_E2P_TESTA;
            break;

        case SM_TESTE_E2P_TESTA:
            tx_string_flash_porta_serial( ( const unsigned char * )"-Gravando...\r\n" );

            ok = 1;
            for( j = 0 ; j < TOTAL_BANCOS_MEMORIA ; j++ )
            {
                #if( EEPROM_TAMANHO == EEPROM_25LC512 )
                    end = ( ( unsigned long )j * 0x10000ul );
                #elif( EEPROM_TAMANHO == EEPROM_25LC1024 )

```



```

        end = ( ( unsigned long )j * 0x20000ul );
    #endif

    for( i = 0 ; i < 100 ; i++ )
    {
        ClrWdt();

        escreve_byte_eeeprom_spi( end + ( unsigned long )end_rnd[ i ], i );
    }

    tx_string_flash_porta_serial( ( const unsigned char * )"-Verificando...\r\n" )
;
    tx_string_flash_porta_serial( ( const unsigned char * )"[Endereco] [Esperado]
[Lido]\r\n" );

    for( j = 0 ; j < TOTAL_BANCOS_MEMORIA ; j++ )
    {
        #if( EEPROM_TAMANHO == EEPROM_25LC512 )
            end = ( ( unsigned long )j * 0x10000ul );
        #elif( EEPROM_TAMANHO == EEPROM_25LC1024 )
            end = ( ( unsigned long )j * 0x20000ul );
        #endif

        for( i = 0 ; i < 100 ; i++ )
        {
            ClrWdt();

            val = le_byte_eeeprom_spi( end + ( unsigned long )end_rnd[ i ] );
            esp = i;

            if( val != esp )
            {
                ok = 0;

                dec_para_ascii( end + ( unsigned long )end_rnd[ i ], buffer_ascii

);

                tx_byte_porta_serial( buffer_ascii[ 0 ] );
                tx_byte_porta_serial( buffer_ascii[ 1 ] );
                tx_byte_porta_serial( buffer_ascii[ 2 ] );
                tx_byte_porta_serial( buffer_ascii[ 3 ] );
                tx_byte_porta_serial( buffer_ascii[ 4 ] );
                tx_byte_porta_serial( buffer_ascii[ 5 ] );
                tx_byte_porta_serial( buffer_ascii[ 6 ] );
                tx_byte_porta_serial( buffer_ascii[ 7 ] );
                tx_byte_porta_serial( ' ' );
                tx_byte_porta_serial( ' ' );
                tx_byte_porta_serial( ' ' ); dec_para_ascii(
                esp, buffer_ascii ); tx_byte_porta_serial(
                buffer_ascii[ 3 ] ); tx_byte_porta_serial(
                buffer_ascii[ 4 ] ); tx_byte_porta_serial(
                buffer_ascii[ 5 ] ); tx_byte_porta_serial(
                buffer_ascii[ 6 ] ); tx_byte_porta_serial(
                buffer_ascii[ 7 ] ); tx_byte_porta_serial( '
                ' ); tx_byte_porta_serial( ' ' );
                tx_byte_porta_serial( ' ' );
                tx_byte_porta_serial( ' ' );
                tx_byte_porta_serial( ' ' );
                tx_byte_porta_serial( ' ' ); dec_para_ascii(
                val, buffer_ascii ); tx_byte_porta_serial(
                buffer_ascii[ 3 ] ); tx_byte_porta_serial(
                buffer_ascii[ 4 ] ); tx_byte_porta_serial(
                buffer_ascii[ 5 ] ); tx_byte_porta_serial(
                buffer_ascii[ 6 ] ); tx_byte_porta_serial(
                buffer_ascii[ 7 ] ); tx_byte_porta_serial(
                '\r' ); tx_byte_porta_serial( '\n' );
            }
        }
    }

    if( ok )
    {

```

```

        tx_string_flash_porta_serial( ( const unsigned char * )"E2P OK\r\n" );
    }
    else
    {
        tx_string_flash_porta_serial( ( const unsigned char * )"E2P ERRO\r\n" );
    }

    // Zera a EEPROM
    for( j = 0 ; j < TOTAL_BANCOS_MEMORIA ; j++ )
    {
        apaga_eeeprom_spi( j );
    }

    sm = SM_TESTE_E2P_INICIA;

    return( 0 );
}

return( 1 );
#endif

return( 0 );
}
#endif

/*****\
* teste_eeeprom_i2c *
* Escreve todos os bytes da EEPROM da memória com o valores de 0 a 255. Ao final da es- *
* crita, lê byte a byte e compara se o valor lido é igual ao esperado. Caso seja dife- *
* rente, mostra endereço, valor esperado e valor lido *
* *
* Parâmetros: void *
* Retorno : 1 teste em andamento, 0 teste finalizado *
\*****/ #if(
TIPO_SW == SW_TESTE )
inline unsigned char teste_eeeprom_i2c( void )
{
    #if( defined( MOD_EEPROM_I2C ) )
        #include <stdlib.h>

        #define TOTAL_BANCOS_MEMORIA 1

        volatile unsigned long i, j, end;
        unsigned char val;
        unsigned char esp;
        unsigned char ok;
        unsigned char igual;
        unsigned int rnd;
        static unsigned int end_rnd[ 100 ];
        static enum
        {
            SM_TESTE_E2P_INICIA = 0,
            SM_TESTE_E2P_TESTA,
        } sm = SM_TESTE_E2P_INICIA;

        switch( sm )
        {
        case SM_TESTE_E2P_INICIA:
            tx_string_flash_porta_serial( ( const unsigned char * )"r\n\r\nTeste de E2P:\r\n" );

            srand( obtem_tick() );
            for( i = 0 ; i < 100 ; i++ )
            {
                ok = 1;
                do
                {
                    ClrWdt();

                    rnd = ( unsigned int )rand();

                    igual = 1;
                    for( j = 0 ; j < i ; j++ )

```

```

        {
            if( end_rnd[ j ] == rnd )
            {
                ok = 0;
                igual = 0;
                j = i;
            }
        }

        if( igual )
        {
            ok = 1;
        }
    }
    while( !ok );

    end_rnd[ i ] = rnd;
}

sm = SM_TESTE_E2P_TESTA;
break;

case SM_TESTE_E2P_TESTA:
    tx_string_flash_porta_serial( ( const unsigned char * )"-Gravando...\r\n" );

    ok = 1;
    for( j = 0 ; j < TOTAL_BANCOS_MEMORIA ; j++ )
    {
        #if( EEPROM_TAMANHO == EEPROM_25LC512 )
            end = ( ( unsigned long )j * 0x10000ul );
        #elif( EEPROM_TAMANHO == EEPROM_25LC1024 )
            end = ( ( unsigned long )j * 0x20000ul );
        #endif

        for( i = 0 ; i < 100 ; i++ )
        {
            ClrWdt();

            escreve_byte_eeeprom_i2c( end + ( unsigned long )end_rnd[ i ], i );
        }

        tx_string_flash_porta_serial( ( const unsigned char * )"-Verificando...\r\n" );
;
        tx_string_flash_porta_serial( ( const unsigned char * )"[Endereco] [Esperado]
[Lido]\r\n" );

        for( j = 0 ; j < TOTAL_BANCOS_MEMORIA ; j++ )
        {
            #if( EEPROM_TAMANHO == EEPROM_25LC512 )
                end = ( ( unsigned long )j * 0x10000ul );
            #elif( EEPROM_TAMANHO == EEPROM_25LC1024 )
                end = ( ( unsigned long )j * 0x20000ul );
            #endif

            for( i = 0 ; i < 100 ; i++ )
            {
                ClrWdt();

                val = le_byte_eeeprom_i2c( end + ( unsigned long )end_rnd[ i ] );
                esp = i;

                if( val != esp )
                {
                    ok = 0;

                    dec_para_ascii( end + ( unsigned long )end_rnd[ i ], buffer_ascii
);

                    tx_byte_porta_serial( buffer_ascii[ 0 ] );
                    tx_byte_porta_serial( buffer_ascii[ 1 ] );
                    tx_byte_porta_serial( buffer_ascii[ 2 ] );
                    tx_byte_porta_serial( buffer_ascii[ 3 ] );
                    tx_byte_porta_serial( buffer_ascii[ 4 ] );
                    tx_byte_porta_serial( buffer_ascii[ 5 ] );

```

```

        tx_byte_porta_serial( buffer_ascii[ 6 ] );
        tx_byte_porta_serial( buffer_ascii[ 7 ] );
        tx_byte_porta_serial( ' ' );
        tx_byte_porta_serial( ' ' );
        tx_byte_porta_serial( ' ' ); dec_para_ascii(
        esp, buffer_ascii ); tx_byte_porta_serial(
        buffer_ascii[ 3 ] ); tx_byte_porta_serial(
        buffer_ascii[ 4 ] ); tx_byte_porta_serial(
        buffer_ascii[ 5 ] ); tx_byte_porta_serial(
        buffer_ascii[ 6 ] ); tx_byte_porta_serial(
        buffer_ascii[ 7 ] ); tx_byte_porta_serial( '
        ' ); tx_byte_porta_serial( ' ' );
        tx_byte_porta_serial( ' ' );
        tx_byte_porta_serial( ' ' );
        tx_byte_porta_serial( ' ' );
        tx_byte_porta_serial( ' ' ); dec_para_ascii(
        val, buffer_ascii ); tx_byte_porta_serial(
        buffer_ascii[ 3 ] ); tx_byte_porta_serial(
        buffer_ascii[ 4 ] ); tx_byte_porta_serial(
        buffer_ascii[ 5 ] ); tx_byte_porta_serial(
        buffer_ascii[ 6 ] ); tx_byte_porta_serial(
        buffer_ascii[ 7 ] ); tx_byte_porta_serial(
        '\r' ); tx_byte_porta_serial( '\n' );
    }
}

if( ok )
{
    tx_string_flash_porta_serial( ( const unsigned char * )"E2P OK\r\n" );
}
else
{
    tx_string_flash_porta_serial( ( const unsigned char * )"E2P ERRO\r\n" );
}

// Zera a EEPROM
for( j = 0 ; j < TOTAL_BANCOS_MEMORIA ; j++ )
{
    apaga_eeeprom_i2c( j );
}

sm = SM_TESTE_E2P_INICIA;

return( 0 );
}

return( 1 );
#endif

return( 0 );
}
#endif

/*****\
* teste_sram *
* Escreve todos os bytes da SRAM da memória com o valores de 0 a 255. Ao final da es- *
* crita, lê byte a byte e compara se o valor lido é igual ao esperado. Caso seja dife- *
* rente, mostra endereço, valor esperado e valor lido *
* *
* Parâmetros: void *
* Retorno : 1 teste em andamento, 0 teste finalizado *
\*****/ #if(
TIPO_SW == SW_TESTE )
inline unsigned char teste_sram( void )
{
    #if( defined( MOD_SRAM ) )
        volatile unsigned long i;
        unsigned char val;
        unsigned char esp;
        unsigned char ok;
    #endif

```

```

static enum
{
    SM_TESTE_SRAM_INICIA = 0,
    SM_TESTE_SRAM_TESTA,
} sm = SM_TESTE_SRAM_INICIA;

switch( sm )
{
case SM_TESTE_SRAM_INICIA:
    tx_string_flash_porta_serial( ( const unsigned char * )"\r\n\r\nTeste de SRAM:
\r\n" );

    sm = SM_TESTE_SRAM_TESTA;
    break;

case SM_TESTE_SRAM_TESTA:
    tx_string_flash_porta_serial( ( const unsigned char * )"-Gravando...\r\n" );

    ok = 1;
    for( i = 0 ; i < 0x8000 ; i++ )
    {
        ClrWdt();

        escreve_byte_sram( i, i );
    }

    tx_string_flash_porta_serial( ( const unsigned char * )"-Verificando...\r\n" )
;
    tx_string_flash_porta_serial( ( const unsigned char * )"[Endereco] [Esperado]
[Lido]\r\n" );

    i = 0;
    while( i < 0x8000 )
    {
        ClrWdt();

        val = le_byte_sram( i );
        esp = i & 0x000000FF;

        if( val != esp )
        {
            ok = 0;

            dec_para_ascii( i, buffer_ascii );
            tx_byte_porta_serial( buffer_ascii[ 3 ] );
            tx_byte_porta_serial( buffer_ascii[ 4 ] );
            tx_byte_porta_serial( buffer_ascii[ 5 ] );
            tx_byte_porta_serial( buffer_ascii[ 6 ] );
            tx_byte_porta_serial( buffer_ascii[ 7 ] );
            tx_byte_porta_serial( ' ' );
            tx_byte_porta_serial( ' ' );
            tx_byte_porta_serial( ' ' );
            tx_byte_porta_serial( ' ' );
            tx_byte_porta_serial( ' ' );
            tx_byte_porta_serial( ' ' );
            tx_byte_porta_serial( ' ' );
            tx_byte_porta_serial( ' ' );
            tx_byte_porta_serial( ' ' );
            dec_para_ascii(
            esp, buffer_ascii ); tx_byte_porta_serial(
            buffer_ascii[ 3 ] ); tx_byte_porta_serial(
            buffer_ascii[ 4 ] ); tx_byte_porta_serial(
            buffer_ascii[ 5 ] ); tx_byte_porta_serial(
            buffer_ascii[ 6 ] ); tx_byte_porta_serial(
            buffer_ascii[ 7 ] ); tx_byte_porta_serial( '
            ' ); tx_byte_porta_serial( ' ' );
            tx_byte_porta_serial( ' ' );
            tx_byte_porta_serial( ' ' );
            tx_byte_porta_serial( ' ' );
            tx_byte_porta_serial( ' ' );
            dec_para_ascii(
            val, buffer_ascii ); tx_byte_porta_serial(
            buffer_ascii[ 3 ] ); tx_byte_porta_serial(
            buffer_ascii[ 4 ] ); tx_byte_porta_serial(
            buffer_ascii[ 5 ] ); tx_byte_porta_serial(
            buffer_ascii[ 6 ] ); tx_byte_porta_serial(
            buffer_ascii[ 7 ] ); tx_byte_porta_serial(
            '\r' );

```

```

        tx_byte_porta_serial( '\n' );
    }

    i++;
}

if( ok )
{
    tx_string_flash_porta_serial( ( const unsigned char * )"SRAM OK\r\n" );
}
else
{
    tx_string_flash_porta_serial( ( const unsigned char * )"SRAM ERRO\r\n" );
}

sm = SM_TESTE_SRAM_INICIA;

return( 0 );
}

return( 1 );
#endif

return( 0 );
}
#endif

/*****\
* teste_rtc *
* Rotina de teste de RTC: envia pela serial a data e hora lidas do RTC no formato *
* dd/mm/aa hh:MM:ss *
* *
* Parâmetros: void *
* Retorno : 1 teste em andamento, 0 teste finalizado *
*****/ #if(
TIPO_SW == SW_TESTE )
inline unsigned char teste_rtc( void )
{
    #if( defined( MOD_RTC ) )
        union unsigned_char aux;
        static unsigned char ultimo_s = 0xFF;
        static unsigned char tempo;
        static enum
        {
            SM_TESTE_RTC_INICIA = 0,
            SM_TESTE_RTC_TESTA,
        } sm = SM_TESTE_RTC_INICIA;

        switch( sm )
        {
            case SM_TESTE_RTC_INICIA:
                tx_string_flash_porta_serial( ( const unsigned char * )"r\n\r\nTeste de RTC:\r\n" );
                tx_string_flash_porta_serial( ( const unsigned char * )" - Leitura do relógio;\r\n" );
                tx_string_flash_porta_serial( ( const unsigned char * )" - Tempo de teste: 5 segundos.\r\n\r\n" );

                tempo = 10;
                ultimo_s = 0xFF;

                sm = SM_TESTE_RTC_TESTA;
                break;

            case SM_TESTE_RTC_TESTA:
                if( F_500MS )
                {
                    F_500MS = 0;

                    le_rtc();

                    if( ultimo_s != rtc.campo.segundo )
                    {

```

```

        ultimo_s = rtc.campo.segundo;

        aux.value = dec_para_bcd( rtc.campo.dia );
        tx_byte_porta_serial( '0' + aux.nibble_high );
        tx_byte_porta_serial( '0' + aux.nibble_low );
        tx_byte_porta_serial( '/' );

        aux.value = dec_para_bcd( rtc.campo.mes );
        tx_byte_porta_serial( '0' + aux.nibble_high );
        tx_byte_porta_serial( '0' + aux.nibble_low );
        tx_byte_porta_serial( '/' );

        aux.value = dec_para_bcd( rtc.campo.ano );
        tx_byte_porta_serial( '0' + aux.nibble_high );
        tx_byte_porta_serial( '0' + aux.nibble_low );
        tx_byte_porta_serial( ' ' );

        aux.value = dec_para_bcd( rtc.campo.hora );
        tx_byte_porta_serial( '0' + aux.nibble_high );
        tx_byte_porta_serial( '0' + aux.nibble_low );
        tx_byte_porta_serial( ':' );

        aux.value = dec_para_bcd( rtc.campo.minuto );
        tx_byte_porta_serial( '0' + aux.nibble_high );
        tx_byte_porta_serial( '0' + aux.nibble_low );
        tx_byte_porta_serial( ':' );

        aux.value = dec_para_bcd( rtc.campo.segundo );
        tx_byte_porta_serial( '0' + aux.nibble_high );
        tx_byte_porta_serial( '0' + aux.nibble_low );

        tx_byte_porta_serial( '\r' );
        tx_byte_porta_serial( '\n' );
    }

    tempo--;
    if( !tempo )
    {
        sm = SM_TESTE_RTC_INICIA;

        return( 0 );
    }
}
break;
}

return( 1 );
#endif

return( 0 );
}
#endif

/*****\
* teste_teclado *
* Rotina de teste de teclado: envia pela serial o estado das TOTAL_TECLAS teclas do te- *
* clado matricial *
* *
* Parâmetros: void *
* Retorno : void *
*****/ #if(
TIPO_SW == SW_TESTE )
inline unsigned char teste_teclado( void )
{
    unsigned char i;

    #if( defined( MOD_IO ) )
        static unsigned char tempo;
        static enum
        {
            SM_TESTE_TECLADO_INICIA = 0,
            SM_TESTE_TECLADO_TESTA,
        } sm = SM_TESTE_TECLADO_INICIA;
    #endif

```

```

switch( sm )
{
case SM_TESTE_TECLADO_INICIA:
    tx_string_flash_porta_serial( ( const unsigned char * )"\r\n\r\nTeste TECLADO:
\r\n" );
    tx_string_flash_porta_serial( ( const unsigned char * )"- Pressione uma tecla
e veja seu valor;\r\n" );
    tx_string_flash_porta_serial( ( const unsigned char * )"- Envio das teclas a
cada pressionamento;\r\n" );
    tx_string_flash_porta_serial( ( const unsigned char * )"- Tempo de teste: 10
segundos.\r\n\r\n" );

    tempo = 20;

    sm = SM_TESTE_TECLADO_TESTA;
    break;

case SM_TESTE_TECLADO_TESTA:
    // Apenas teclas pressionadas
    for( i = 0 ; i < TOTAL_TECLAS ; i++ )
    {
        if( io.teclado_matriz.tecla[ i ].press )
        {
            tx_byte_porta_serial( 't' );
            tx_byte_porta_serial( 'e' );
            tx_byte_porta_serial( 'c' );
            tx_byte_porta_serial( 'l' );
            tx_byte_porta_serial( 'a' );
            tx_byte_porta_serial( ':' );
            tx_byte_porta_serial( ' ' );
            tx_byte_porta_serial( '0' + ( i / 10 ) );
            tx_byte_porta_serial( '0' + ( i % 10 ) );
            tx_byte_porta_serial( '\r' );
            tx_byte_porta_serial( '\n' );
        }
    }

    if( F_500MS )
    {
        F_500MS = 0;

        tempo--;
        if( !tempo )
        {
            sm = SM_TESTE_TECLADO_INICIA;

            return( 0 );
        }
    }
    break;
}

return( 1 );
#endif

return( 0 );
}
#endif

/*****
* teste_io
* Rotina de teste dos I/O's: envia pela serial o estado de cada uma das entradas e a
* cada base de tempo, aciona uma das saídas e mantém demais desligadas.
*
* Parâmetros: void
* Retorno : 1 teste em andamento, 0 teste finalizado
*****/ #if(
TIPO_SW == SW_TESTE )
inline unsigned char teste_io( void )
{
    #if( defined( MOD_IO ) )
        #define TOTAL_SAIDAS 11

```



```
static unsigned char i;
static unsigned char tempo;
static unsigned char saida = 0;
static unsigned int shift = 0;
static unsigned int *ptr;
static enum
{
    SM_TESTE_IO_INICIA = 0,
    SM_TESTE_IO_TESTA,
} sm = SM_TESTE_IO_INICIA;

switch( sm )
{
case SM_TESTE_IO_INICIA:
    tx_string_flash_porta_serial( ( const unsigned char * )"r\n\nTeste I/Os:r\n" );
    tx_string_flash_porta_serial( ( const unsigned char * )"- Acionamento individual de cada uma das saídas a cada 1s;r\n" );
    tx_string_flash_porta_serial( ( const unsigned char * )"- Envio do estado das entradas a cada 1s;r\n" );
    tx_string_flash_porta_serial( ( const unsigned char * )"- Tempo de teste: 5 segundos.r\n\n" );

    tempo = 5;

    i = 0;
    ptr = ( unsigned int * )&io.saidas;
    shift = 0x0004;

    sm = SM_TESTE_IO_TESTA;
    break;

case SM_TESTE_IO_TESTA:
    if( F_1000MS )
    {
        F_1000MS = 0;

        *ptr &= 0x03 ;
        *ptr |= shift ;
        shift <<= 1;

        tx_byte_porta_serial( 'i' );
        tx_byte_porta_serial( 'n' );
        tx_byte_porta_serial( '0' + ( i / 10 ) );
        tx_byte_porta_serial( '0' + ( i % 10 ) );
        tx_byte_porta_serial( ':' );
        tx_byte_porta_serial( 'r' );
        tx_byte_porta_serial( '\n' );
        i++;

        saida++;
        if( saida == TOTAL_SAIDAS )
        {
            saida = 0;
        }

        tempo--;
        if( !tempo )
        {
            sm = SM_TESTE_IO_INICIA;

            return( 0 );
        }
    }
    break;
}

return( 1 );
#endif

return( 0 );
}
#endif
```

```

/*****\
* teste_adc *
* Rotina de teste dos AD's: envia pela serial o valor das conversões A/D, já em ASCII *
* *
* Parâmetros: void *
* Retorno : void *
\*****/ #if(
TIPO_SW == SW_TESTE )
inline unsigned char teste_adc( void )
{
    #if( defined( MOD_ADC ) )
        unsigned char i;

        static unsigned char tempo;
        static enum
        {
            SM_TESTE_ADC_INICIA = 0,
            SM_TESTE_ADC_TESTA,
        } sm = SM_TESTE_ADC_INICIA;

        switch( sm )
        {
            case SM_TESTE_ADC_INICIA:
                tx_string_flash_porta_serial( ( const unsigned char * )"\r\n\r\nTeste ADC:\r\n
                " );
                tx_string_flash_porta_serial( ( const unsigned char * )"- Leitura de todos os
                canais a cada 1s;\r\n" );
                tx_string_flash_porta_serial( ( const unsigned char * )"- Tempo de teste: 5
                segundos.\r\n\r\n" );

                tempo = 5;

                sm = SM_TESTE_ADC_TESTA;
                break;

            case SM_TESTE_ADC_TESTA:
                if( F_1000MS )
                {
                    F_1000MS = 0;

                    for( i = 0 ; i < ADC_NUM_CANAIS ; i++ )
                    {
                        dec_para_ascii( adc.dados.canal[ i ], buffer_ascii );
                        tx_byte_porta_serial( 'a' );
                        tx_byte_porta_serial( 'd' );
                        tx_byte_porta_serial( 'c' );
                        tx_byte_porta_serial( '[' );
                        tx_byte_porta_serial( '0' + i );
                        tx_byte_porta_serial( ']' );
                        tx_byte_porta_serial( ':' );
                        tx_byte_porta_serial( ' ' );
                        tx_byte_porta_serial( buffer_ascii[ 4 ] );
                        tx_byte_porta_serial( buffer_ascii[ 5 ] );
                        tx_byte_porta_serial( buffer_ascii[ 6 ] );
                        tx_byte_porta_serial( buffer_ascii[ 7 ] );
                        tx_byte_porta_serial( '\r' );
                        tx_byte_porta_serial( '\n' );
                    }

                    tempo--;
                    if( !tempo )
                    {
                        sm = SM_TESTE_ADC_INICIA;

                        return( 0 );
                    }
                }
                break;
        }

        return( 1 );
    #endif
}

```

```

    return( 0 );
}
#endif

/*****\
* teste_ethernet *
* Rotina de teste do módulo ethernet - link *
* *
* Parâmetros: void *
* Retorno : void *
\*****/ #if(
TIPO_SW == SW_TESTE )
inline unsigned char teste_ethernet( void )
{
    #if( defined( MOD_ETHERNET ) )
        static unsigned char tempo;
        static enum
        {
            SM_TESTE_ETHERNET_INICIA = 0,
            SM_TESTE_ETHERNET_TESTA,
        } sm = SM_TESTE_ETHERNET_INICIA;

        switch( sm )
        {
        case SM_TESTE_ETHERNET_INICIA:
            salva_dados_default_ethernet();
            inicializa_ethernet();

            tx_string_flash_porta_serial( ( const unsigned char * )"\r\n\r\nTeste Ethernet
:\r\n" );
            tx_string_flash_porta_serial( ( const unsigned char * )"- Tempo de teste: 2
minutos.\r\n" );

            tempo = 120;

            sm = SM_TESTE_ETHERNET_TESTA;
            break;

        case SM_TESTE_ETHERNET_TESTA:
            executa_ethernet();

            if( F_1000MS )
            {
                F_1000MS = 0;

                tempo--;
                if( !tempo )
                {
                    sm = SM_TESTE_ETHERNET_INICIA;

                    return( 0 );
                }
            }
            break;
        }

        return( 1 );
    #endif
}

return( 0 );
#endif

/*****\
* teste_gprs *
* Rotina de teste módulo GPRS: inicialização do módulo, conexão à rede GPRS, obtenção *
* de IP da operadora e conexão TCP/IP com um servidor remoto *
* *
* Parâmetros: void *
* Retorno : 1 teste em andamento, 0 teste finalizado *
\*****/

```

```

#if( TIPO_SW == SW_TESTE )
inline unsigned char teste_gprs( void )
{
    #if( defined( MOD_GPRS ) )
        const unsigned char IP_SVR[] = "\xC8\xCF\x4B\x33"; // 200.207.75.51
        const unsigned int PORTA_SVR = 5031;

        // Definir as flags de software (estas flags serão utilizadas para controle da
        aplicação):
        union unsigned_char fs_sw;
        #define FS_FALHA_CONEXAO_SVR fs_sw.bit0 // Usada pela aplicação para saber
        quando houve alguma falha de conexão
        #define FS_CONECTA_SVR fs_sw.bit1 // Usada pela aplicação para realizar
        uma conexão com o servidor
        #define FS_CONEXAO_SVR fs_sw.bit2 // Usada pela aplicação para saber
        quando a conexão com o servidor foi estabelecida com sucesso
        #define FS_TX_DADOS_SVR fs_sw.bit3 // Após a aplicação carregar o buffer
        msg_grps, ela seta esta flag para realizar a transmissão
        #define FS_DADOS_RX_SVR fs_sw.bit4 // Usada pela aplicação testar se foi
        recebido algo no buffer msg_grps

        static unsigned long t_ini;
        static struct
        {
            unsigned char gprs_on : 1;
            unsigned char : 7;
        } flags = { 0x00 };
        static enum
        {
            TESTE_GPRS_OCIOSO = 0,
            TESTE_GPRS_INICIA,
            TESTE_GPRS_LIGA_MODULO,
            TESTE_GPRS_AGUARDA_CONEXAO_REDE,
            TESTE_GPRS_CONECTA_SERVIDOR,
            TESTE_GPRS_AGUARDA_CONEXAO_SERVIDOR,
            TESTE_GPRS_AGUARDA_SOCKET,
            TESTE_GPRS_INICIALIZA_BUFFERS,
            TESTE_GPRS_COMUNICACAO,
            TESTE_GPRS_TX_DADOS,
            TESTE_GPRS_AGUARDA_TX_DADOS,
            TESTE_GPRS_DESCONECTA_SERVIDOR,
            TESTE_GPRS_AGUARDA_DESCONEXAO_SERVIDOR,
            TESTE_GPRS_DESLIGA_MODULO,
            TESTE_GPRS_DESLIGADO,
        } sm = TESTE_GPRS_OCIOSO;

        executa_gprs();

        // Verifica estado da conexão com o servidor TCP. Caso servidor tenha caído, deve
        reiniciar máquina de estados
        if( ( sm >= TESTE_GPRS_COMUNICACAO ) &&
            ( sm < TESTE_GPRS_DESCONECTA_SERVIDOR ) )
        {
            if( ( !conexao_ok_gprs() ) ||
                ( !socket_ok_gprs() ) )
            {
                FS_FALHA_CONEXAO_SVR = 1;

                sm = TESTE_GPRS_DESLIGADO;
            }
        }
        else if( ( sm >= TESTE_GPRS_AGUARDA_CONEXAO_REDE ) && (
            sm <= TESTE_GPRS_AGUARDA_SOCKET ) )
        {
            if( tempo_decorrido( t_ini, T_X1SEG ) > T_MAX_AGUARDA_CONEXAO_SVR )
            {
                FS_FALHA_CONEXAO_SVR = 1;

                sm = TESTE_GPRS_DESLIGADO;
            }
        }
        if( falha_geral_gprs() )
        {

```

```
    desliga_gprs();
    flags.gprs_on = 0;

    FS_FALHA_CONEXAO_SVR = 1;

    sm = TESTE_GPRS_DESLIGADO;
}

switch( sm )
{
case TESTE_GPRS_OCIOSO:
    if( FS_CONECTA_SVR )
    {
        FS_FALHA_CONEXAO_SVR = 0;

        sm = TESTE_GPRS_INICIA;
    }
    break;

case TESTE_GPRS_INICIA:
    t_ini = obtem_segundo_atual();

    sm = TESTE_GPRS_LIGA_MODULO;
    break;

case TESTE_GPRS_LIGA_MODULO:
    configura_operadora_gprs( setup.cco.apn, setup.cco.login, setup.cco.senha );

    if( !flags.gprs_on )
    {
        liga_gprs();
    }

    sm = TESTE_GPRS_AGUARDA_CONEXAO_REDE;
    break;

case TESTE_GPRS_AGUARDA_CONEXAO_REDE:
    if( status_rede_gprs() == REDE_CONECTADO_OK )
    {
        if( !conexao_ok_gprs() )
        {
            flags.gprs_on = 1;

            sm = TESTE_GPRS_CONECTA_SERVIDOR;
        }
    }
    break;

case TESTE_GPRS_CONECTA_SERVIDOR:
    configura_servidor_gprs( setup.cco.ip, setup.cco.porta );

    conecta_servidor_gprs();

    sm = TESTE_GPRS_AGUARDA_CONEXAO_SERVIDOR;
    break;

case TESTE_GPRS_AGUARDA_CONEXAO_SERVIDOR:
    if( conexao_ok_gprs() )
    {
        sm = TESTE_GPRS_AGUARDA_SOCKET;
    }
    break;

case TESTE_GPRS_AGUARDA_SOCKET:
    if( socket_ok_gprs() )
    {
        FS_CONEXAO_SVR = 1;

        sm = TESTE_GPRS_INICIALIZA_BUFFERS;
    }
    break;

case TESTE_GPRS_INICIALIZA_BUFFERS:
    limpa_buffer_gprs();
```

```
FS_TX_DADOS_SVR = 0;
FS_DADOS_RX_SVR = 0;

// Começa a receber dados do servidor TCP (realiza um loopback, devolvendo os
dados para o servidor)
sm = TESTE_GPRS_COMUNICACAO;
break;

case TESTE_GPRS_COMUNICACAO:
// Verificação de pacote recebido
if( !FS_CONECTA_SVR )
{
    sm = TESTE_GPRS_DESCONECTA_SERVIDOR;
}
else if( msg_tcp_recebida_gprs() )
{
    // IMPORTANTE: reseta buffer de recepção e flag de pacote recebido
    limpa_buffer_gprs();

    /*** CAMADA DE APLICAÇÃO DE PROTOCOLOS: Recepção de dados ***/
    {
        FS_DADOS_RX_SVR = 1;
    }
}
else if( FS_TX_DADOS_SVR )
{
    sm = TESTE_GPRS_TX_DADOS;
}
break;

case TESTE_GPRS_TX_DADOS:
/*** CAMADA DE APLICAÇÃO DE PROTOCOLOS: Transmissão de dados ***/
{
    envia_msg_tcp_gprs();

    sm = TESTE_GPRS_AGUARDA_TX_DADOS;
}
break;

case TESTE_GPRS_AGUARDA_TX_DADOS:
/*** CAMADA DE APLICAÇÃO DE PROTOCOLOS: Transmissão de dados ***/
if( msg_tcp_enviada_gprs() )
{
    FS_TX_DADOS_SVR = 0;

    sm = TESTE_GPRS_COMUNICACAO;
}
break;

case TESTE_GPRS_DESCONECTA_SERVIDOR:
desconecta_servidor_gprs();

sm = TESTE_GPRS_AGUARDA_DESCONEXAO_SERVIDOR;
break;

case TESTE_GPRS_AGUARDA_DESCONEXAO_SERVIDOR:
if( !conexao_ok_gprs() )
{
    sm = TESTE_GPRS_DESLIGA_MODULO;
}
break;

case TESTE_GPRS_DESLIGA_MODULO:
// Se a aplicação necessitar desenergizar o módulo, descomente as duas linhas
abaixo
/*
desliga_gprs();
flags.gprs_on = 0;
*/

FS_CONECTA_SVR = 0;

sm = TESTE_GPRS_DESLIGADO;
```

```

        break;

    case TESTE_GPRS_DESLIGADO:
        FS_CONEXAO_SVR = 0;

        sm = TESTE_GPRS_OCIOSO;
        return( 1 );
    }

    return( 0 );
#endif

return( 0 );
#endif
}

/*****\
* teste_uart *
* Rotina de teste de UART: envia pela serial qualquer os dados que foram recebidos da *
* mesma (loopback), porém com os caracteres em letra maiúscula; *
* *
* Parâmetros: void *
* Retorno : 1 teste em andamento, 0 teste finalizado *
\*****/ #if(
TIPO_SW == SW_TESTE )
inline unsigned char teste_uart( void )
{
    #if( defined( MOD_UART ) )
        static unsigned char tempo;
        static enum
        {
            SM_TESTE_UART_INICIA = 0,
            SM_TESTE_UART_TESTA,
        } sm = SM_TESTE_UART_INICIA;

        switch( sm )
        {
        case SM_TESTE_UART_INICIA:
            tx_string_flash_porta_serial( ( const unsigned char * )"r\nr\nTeste UART\r\n
" );
            tx_string_flash_porta_serial( ( const unsigned char * )"Loopback;\r\n" );
            tx_string_flash_porta_serial( ( const unsigned char * )"Tempo de teste: 5
segundos.\r\n\r\n" );

            tempo = 5;

            sm = SM_TESTE_UART_TESTA;
            break;

        case SM_TESTE_UART_TESTA:
            if( F_PORTA_SERIAL_PCT_RX )
            {
                for( porta_serial.tam_tx = 0 ; porta_serial.tam_tx < porta_serial.tam_rx ;
porta_serial.tam_tx++ )
                {
                    porta_serial.tx[ porta_serial.tam_tx ] = upper_chr( porta_serial.rx[
porta_serial.tam_tx ] );
                }
                tx_pacote_porta_serial();

                porta_serial.tam_rx = 0;

                F_PORTA_SERIAL_PCT_RX = 0;
            }

            if( F_1000MS )
            {
                F_1000MS = 0;

                tempo--;
                if( !tempo )
                {
                    sm = SM_TESTE_UART_INICIA;

```

```

        return( 0 );
    }
    }
    break;
}

return( 1 );
#endif

return( 0 );
}
#endif

/*****\
* teste_rtc *
* Rotina de teste de RTC: envia pela serial a data e hora lidas do RTC no formato *
* dd/mm/aa hh:MM:ss . *
* *
* Parâmetros: void *
* Retorno : 1 teste em andamento, 0 teste finalizado *
\*****/ #if(
TIPO_SW == SW_TESTE )
inline unsigned char teste_rtc( void )
{
    #if( defined( MOD_RTC ) )
        union unsigned_char aux;
        static unsigned char ultimo_s = 0xFF;
        static unsigned char tempo;
        static enum
        {
            SM_TESTE_RTC_INICIA = 0,
            SM_TESTE_RTC_TESTA,
        } sm = SM_TESTE_RTC_INICIA;

        switch( sm )
        {
        case SM_TESTE_RTC_INICIA:
            tx_string_flash_porta_serial( ( const unsigned char * )"r\n\r\nTeste de RTC:\n" );
            tx_string_flash_porta_serial( ( const unsigned char * )"- Leitura do relógio;\n" );
            tx_string_flash_porta_serial( ( const unsigned char * )"- Tempo de teste: 5 segundos.\n\r\n" );

            tempo = 10;
            ultimo_s = 0xFF;

            sm = SM_TESTE_RTC_TESTA;
            break;

        case SM_TESTE_RTC_TESTA:
            if( F_500MS )
            {
                F_500MS = 0;

                le_rtc();

                if( ultimo_s != rtc.campo.segundo )
                {
                    ultimo_s = rtc.campo.segundo;

                    aux.value = dec_para_bcd( rtc.campo.dia );
                    tx_byte_porta_serial( '0' + aux.nibble_high );
                    tx_byte_porta_serial( '0' + aux.nibble_low );
                    tx_byte_porta_serial( '/' );

                    aux.value = dec_para_bcd( rtc.campo.mes );
                    tx_byte_porta_serial( '0' + aux.nibble_high );
                    tx_byte_porta_serial( '0' + aux.nibble_low );
                    tx_byte_porta_serial( '/' );

                    aux.value = dec_para_bcd( rtc.campo.ano );

```



```

        tx_byte_porta_serial( '0' + aux.nibble_high );
        tx_byte_porta_serial( '0' + aux.nibble_low );
        tx_byte_porta_serial( ' ' );

        aux.value = dec_para_bcd( rtc.campo.hora );
        tx_byte_porta_serial( '0' + aux.nibble_high );
        tx_byte_porta_serial( '0' + aux.nibble_low );
        tx_byte_porta_serial( ':' );

        aux.value = dec_para_bcd( rtc.campo.minuto );
        tx_byte_porta_serial( '0' + aux.nibble_high );
        tx_byte_porta_serial( '0' + aux.nibble_low );
        tx_byte_porta_serial( ':' );

        aux.value = dec_para_bcd( rtc.campo.segundo );
        tx_byte_porta_serial( '0' + aux.nibble_high );
        tx_byte_porta_serial( '0' + aux.nibble_low );

        tx_byte_porta_serial( '\r' );
        tx_byte_porta_serial( '\n' );
    }

    tempo--;
    if( !tempo )
    {
        sm = SM_TESTE_RTC_INICIA;

        return( 0 );
    }
}
break;
}
return( 1 );
#endif

return( 0 );
}
#endif

/*****\
* teste_motor *
* Rotina de teste do motor de infusao, liga por 5 segundos e depois desliga *
* Esta rotina deve ser chamada em uma base de tempo de 1000ms *
* *
* Parâmetros: void *
* Retorno : void *
*****/ #if(
TIPO_SW == SW_TESTE )
inline unsigned char teste_motor( void )
{
    static unsigned char tempo;
    static unsigned int passos_acc;
    static unsigned int encl_acc;
    static unsigned int enc2_acc;
    unsigned char i = 0;
    static enum
    {
        SM_TESTE_MOTOR_INICIA = 0,
        SM_TESTE_MOTOR_TESTA,
    } sm = SM_TESTE_MOTOR_INICIA;

    switch( sm )
    {
        case SM_TESTE_MOTOR_INICIA:
            tx_string_flash_porta_serial( ( const unsigned char * )" \r \n \r \n Teste MOTOR \r \n " );
            tx_string_flash_porta_serial( ( const unsigned char * )" vazao= 1000ml/h Teste 5s - Tx/1000passos \r \n " );

            tempo = 5;
            passos_acc = 0;
            encl_acc = 0;

```

```
enc2_acc    = 0;

dados.info.processo = PROCESSO_EM_ANDAMENTO;
dados.parametros.vazao = 10000;
inicializa_infusao( SENTIDO_AVANCO, VOLUME_CONTINUO );

sm = SM_TESTE_MOTOR_TESTA;
break;

case SM_TESTE_MOTOR_TESTA:
if( F_50MS )
{
    F_50MS = 0;

    passos_acc += motor.passos_dados;
    enc1_acc    += motor.encoder[0].valor;
    enc2_acc    += motor.encoder[1].valor;

    if( passos_acc >= 1000 )
    {
        i= 0;
        dec_para_ascii( enc1_acc, buffer_ascii );
        tx_byte_porta_serial( 'e' );
        tx_byte_porta_serial( 'n' );
        tx_byte_porta_serial( 'c' );
        tx_byte_porta_serial( ' ' );
        tx_byte_porta_serial( '1' );
        tx_byte_porta_serial( ':' );
        tx_byte_porta_serial( buffer_ascii[ 4 ] );
        tx_byte_porta_serial( buffer_ascii[ 5 ] );
        tx_byte_porta_serial( buffer_ascii[ 6 ] );
        tx_byte_porta_serial( buffer_ascii[ 7 ] );
        tx_byte_porta_serial( '\r' );
        tx_byte_porta_serial( '\n' );

        dec_para_ascii( enc2_acc, buffer_ascii );
        tx_byte_porta_serial( 'e' );
        tx_byte_porta_serial( 'n' );
        tx_byte_porta_serial( 'c' );
        tx_byte_porta_serial( ' ' );
        tx_byte_porta_serial( '2' );
        tx_byte_porta_serial( ':' );
        tx_byte_porta_serial( buffer_ascii[ 4 ] );
        tx_byte_porta_serial( buffer_ascii[ 5 ] );
        tx_byte_porta_serial( buffer_ascii[ 6 ] );
        tx_byte_porta_serial( buffer_ascii[ 7 ] );
        tx_byte_porta_serial( '\r' );
        tx_byte_porta_serial( '\n' );

        dec_para_ascii( adc.dados.canais.s_bolhas,    buffer_ascii );
        tx_byte_porta_serial( 'b' );
        tx_byte_porta_serial( 'o' );
        tx_byte_porta_serial( 'l' );
        tx_byte_porta_serial( 'h' );
        tx_byte_porta_serial( 'a' );
        tx_byte_porta_serial( ':' );
        tx_byte_porta_serial( buffer_ascii[ 4 ] );
        tx_byte_porta_serial( buffer_ascii[ 5 ] );
        tx_byte_porta_serial( buffer_ascii[ 6 ] );
        tx_byte_porta_serial( buffer_ascii[ 7 ] );
        tx_byte_porta_serial( '\r' );
        tx_byte_porta_serial( '\n' );

        dec_para_ascii( adc.dados.canais.s_force_b, buffer_ascii );
        tx_byte_porta_serial( 'f' );
        tx_byte_porta_serial( 'o' );
        tx_byte_porta_serial( 'r' );
        tx_byte_porta_serial( 'c' );
        tx_byte_porta_serial( 'B' );
        tx_byte_porta_serial( ':' );
        tx_byte_porta_serial( buffer_ascii[ 4 ] );
        tx_byte_porta_serial( buffer_ascii[ 5 ] );
        tx_byte_porta_serial( buffer_ascii[ 6 ] );
        tx_byte_porta_serial( buffer_ascii[ 7 ] );
    }
}
```

```
tx_byte_porta_serial( '\r' );
tx_byte_porta_serial( '\n' );

dec_para_ascii( adc.dados.canais.s_force_a, buffer_ascii );
tx_byte_porta_serial( 'f' );
tx_byte_porta_serial( 'o' );
tx_byte_porta_serial( 'r' );
tx_byte_porta_serial( 'c' );
tx_byte_porta_serial( 'A' );
tx_byte_porta_serial( ':' );
tx_byte_porta_serial( buffer_ascii[ 4 ] );
tx_byte_porta_serial( buffer_ascii[ 5 ] );
tx_byte_porta_serial( buffer_ascii[ 6 ] );
tx_byte_porta_serial( buffer_ascii[ 7 ] );
tx_byte_porta_serial( '\n' );
tx_byte_porta_serial( '\r' );

dec_para_ascii( adc.dados.canais.m_i_ph1, buffer_ascii );
tx_byte_porta_serial( 'i' );
tx_byte_porta_serial( '-' );
tx_byte_porta_serial( 'p' );
tx_byte_porta_serial( 'h' );
tx_byte_porta_serial( '1' );
tx_byte_porta_serial( ':' );
tx_byte_porta_serial( buffer_ascii[ 4 ] );
tx_byte_porta_serial( buffer_ascii[ 5 ] );
tx_byte_porta_serial( buffer_ascii[ 6 ] );
tx_byte_porta_serial( buffer_ascii[ 7 ] );
tx_byte_porta_serial( '\r' );
tx_byte_porta_serial( '\n' );

dec_para_ascii( adc.dados.canais.m_i_ph2, buffer_ascii );
tx_byte_porta_serial( 'i' );
tx_byte_porta_serial( '-' );
tx_byte_porta_serial( 'p' );
tx_byte_porta_serial( 'h' );
tx_byte_porta_serial( '2' );
tx_byte_porta_serial( ':' );
tx_byte_porta_serial( buffer_ascii[ 4 ] );
tx_byte_porta_serial( buffer_ascii[ 5 ] );
tx_byte_porta_serial( buffer_ascii[ 6 ] );
tx_byte_porta_serial( buffer_ascii[ 7 ] );
tx_byte_porta_serial( '\r' );
tx_byte_porta_serial( '\n' );
tx_byte_porta_serial( '\r' );
tx_byte_porta_serial( '\n' );
if( FF_ERRO_REDUNDANCIA )
{
    FF_ERRO_REDUNDANCIA = 0;
    tx_byte_porta_serial( ' ' );
    tx_byte_porta_serial( 'E' );
}
tx_byte_porta_serial( '\r' );
tx_byte_porta_serial( '\n' );

passos_acc = 0;
encl1_acc  = 0;
enc2_acc   = 0;
}
}

if( F_1000MS )
{
    F_1000MS = 0;

    tempo--;
    if( !tempo )
    {
        finaliza_infusao();

        sm = SM_TESTE_MOTOR_INICIA;

        return( 0 );
    }
}
```

```
        }
        break;
    }
    return( 1 );
}
#endif
```