

Project # eZ2932

PolyMoniCon

A multiple machine monitor and controller with easy-to-use web browser user interface

ABSTRACT

Network remote control is becoming commonplace for all kinds of devices these days. Routers, alarm systems, even refrigerators are being equipped with browser based interfaces for control, configuration and monitoring.

ZiLOG obviously had those kinds of applications in mind for its new generation of eZ80 Acclaim! Microcontrollers, which possess an impressive range of features for adding network functionality to new and existing designs.

The application I had in mind was monitoring a number of machines in an amusement arcade. I wanted to be able to assess the throughput of coins for each machine and view the information using a PC with a standard Internet browser.

At first I considered embedding an eZ80 E-NET module in each machine, but I soon realised that that would be overkill, and besides, it would also require running a LAN cable to each machine: Impractical as the machines are frequently changed or moved.

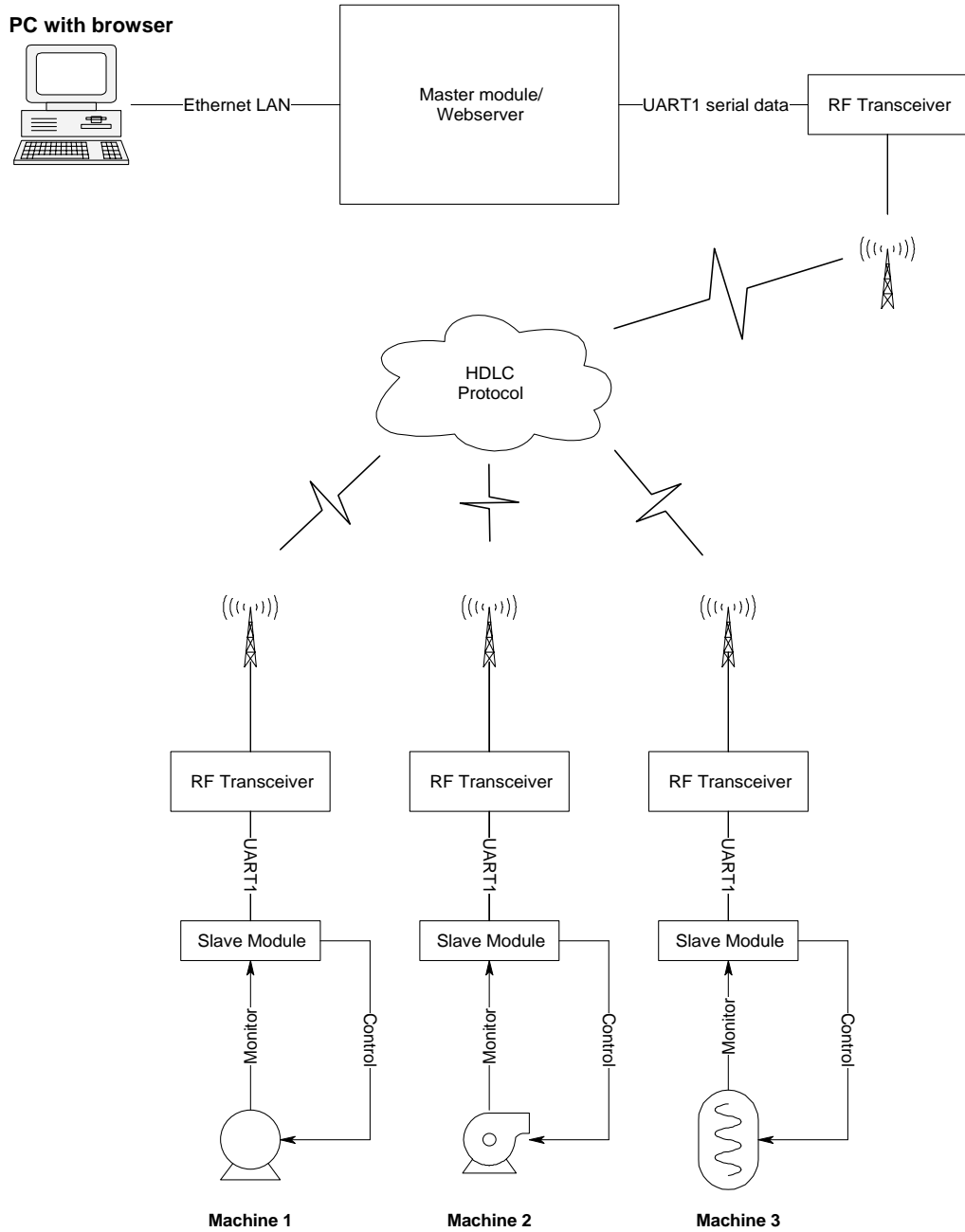
I reasoned that a wireless solution would be best. I also realised that a ZiLOG eZ8 Encore! Microcontroller would be the perfect complement to the eZ80 module in providing a machine interface. The eZ8 would be a slave reporting to the eZ80 master, which would provide the web interface.

An RF data transceiver fitted to the master and each slave allows communication between all the devices. A customised HDLC protocol is used to facilitate the transfer of data, and provides a high degree of error tolerance and recovery necessary when using RF communications.

Up to 127 slaves can be monitored by the master. The slave modules connect directly to the machine being monitored and provide four counter inputs for counting pulse signals (e.g. coin switch), three control outputs, and eight analog inputs, which may be used for measuring voltage, temperature sensors, etc. All sorts of applications are possible; Home automation, process monitoring, and alarm control just to name a few.

Welcome to PolyMoniCon! 'Poly' meaning many, and 'MoniCon' short for Monitoring and Control.

BLOCK DIAGRAM



PHOTOS



Photo 1. Slave module

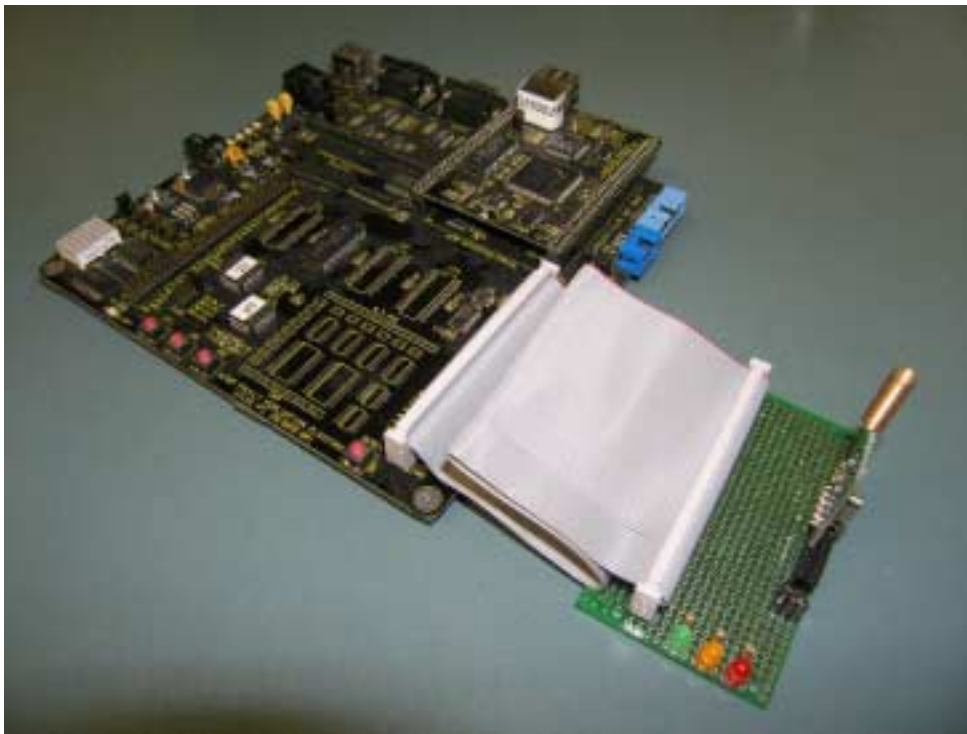


Photo 2. Master module

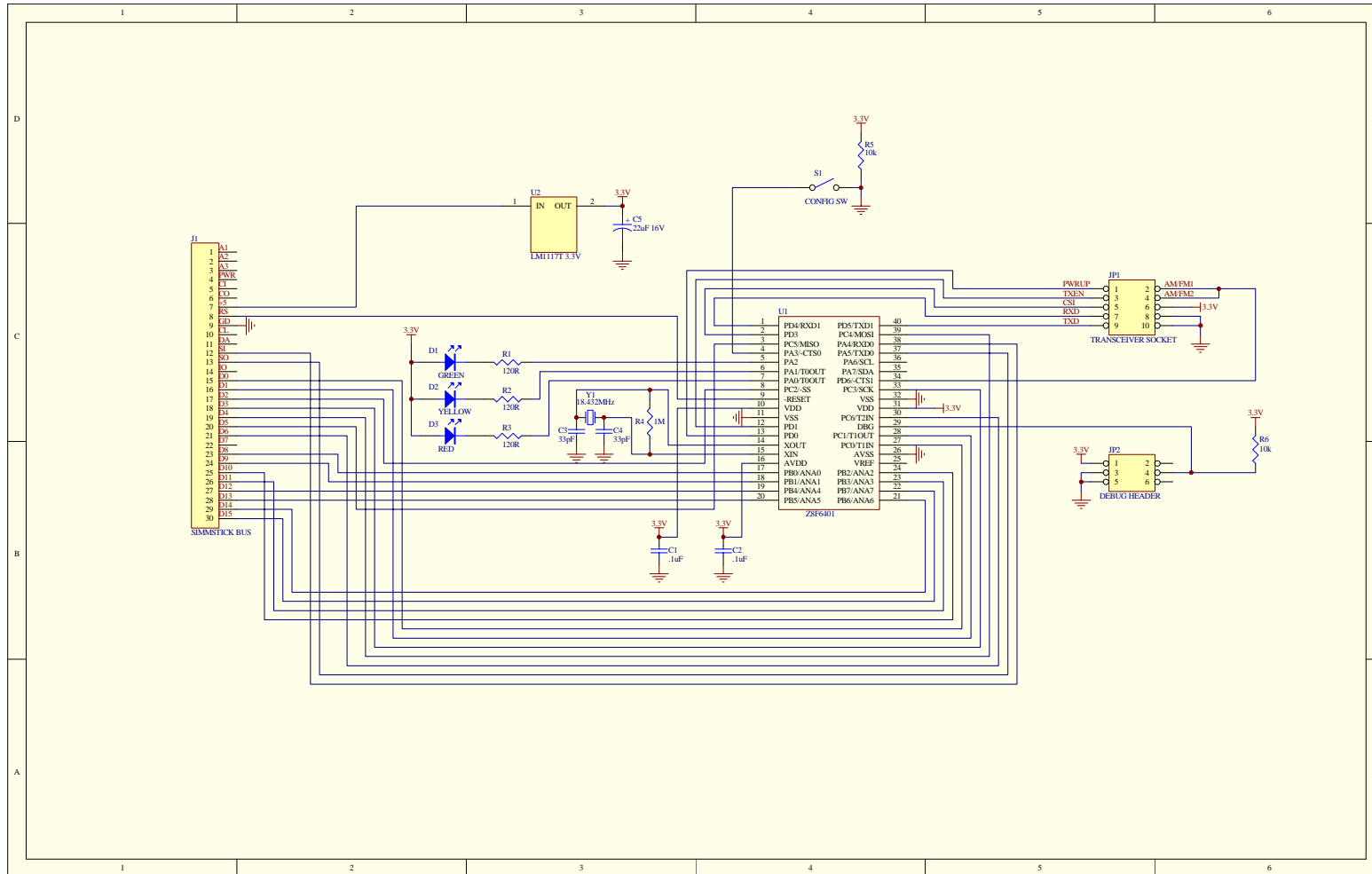


Figure 1. Slave module schematic

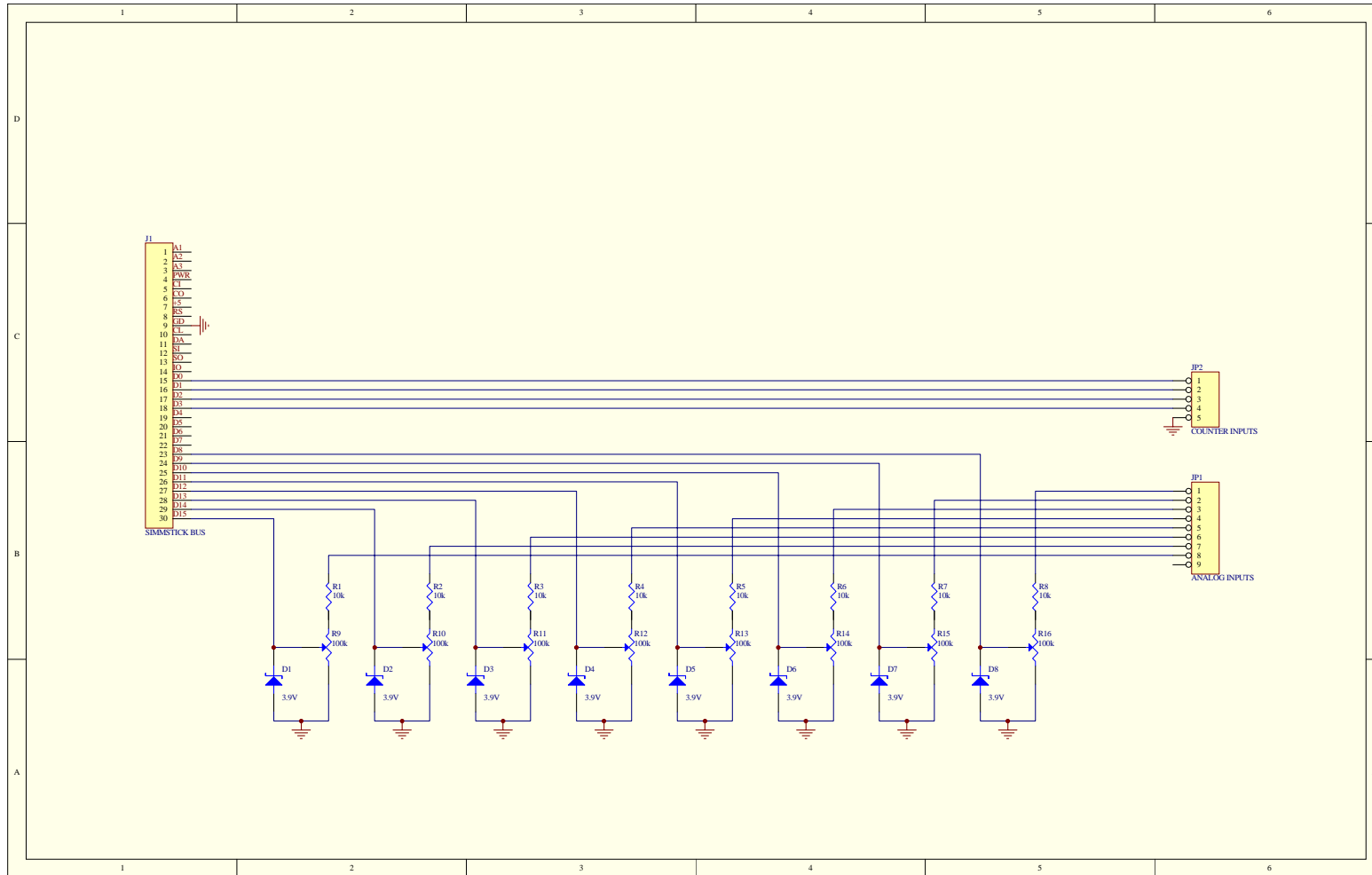


Figure 2. Slave input schematic

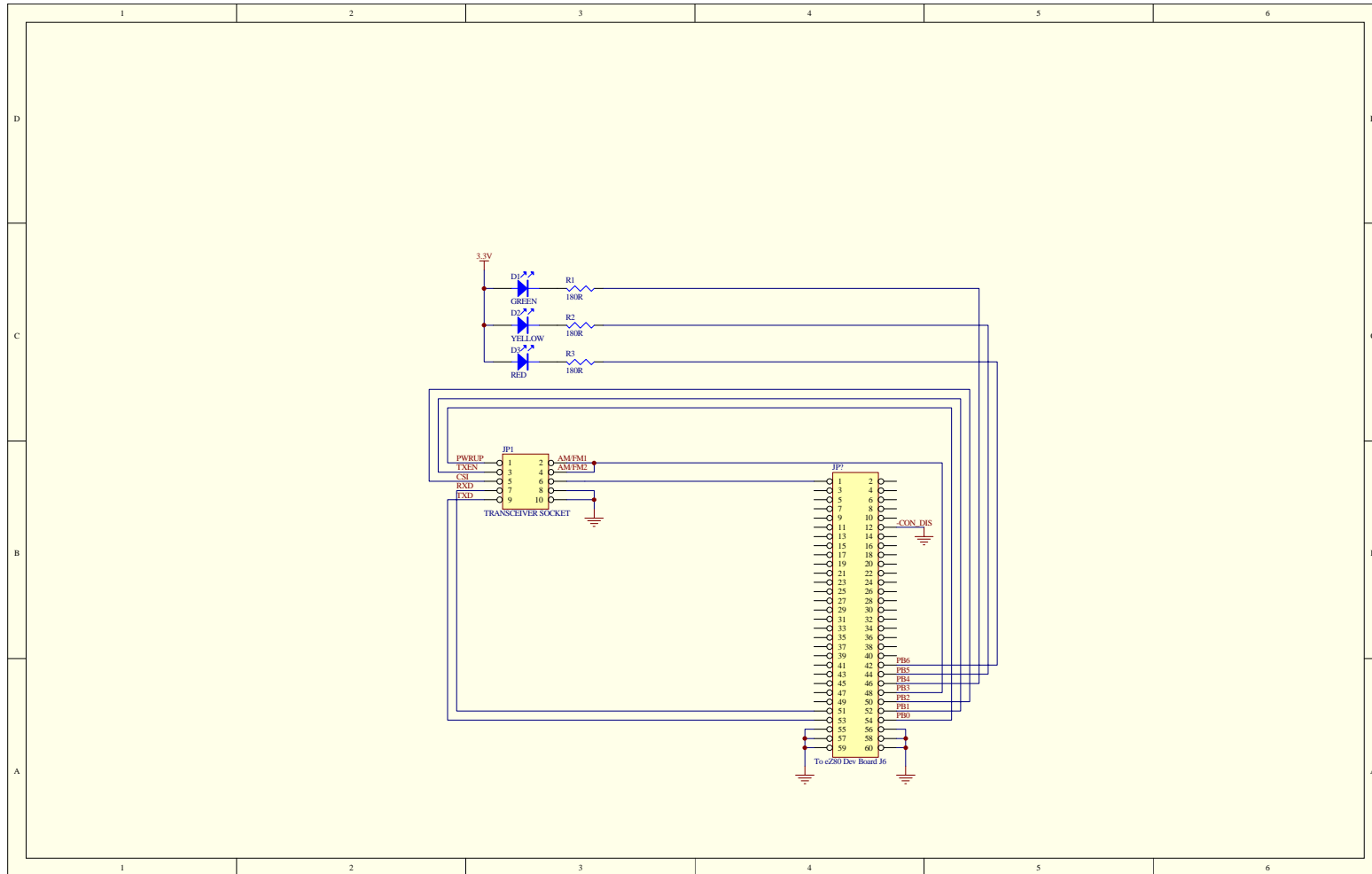


Figure 3. Master node transceiver interface schematic

CODE SAMPLE (hdlc.h)

```
#ifndef _HDLC_H_
#define _HDLC_H_

// Maximum size of information field in HDLC frame
#define HDLC_MAX_INFO_SIZE      80

// Maximum number of buffers
#define HDLC_MAX_BUFFERS       8

// HDLC frame structure
typedef struct HDLCFRAME_S
{
    unsigned char    address;
    unsigned char    control;
    char             info[HDLC_MAX_INFO_SIZE + sizeof(unsigned short)];
    unsigned char    length;
} HDLCFRAME_S;

// HDLC secondary structure
typedef struct HDLCSECONDARY_S
{
    unsigned char    mode;
    unsigned char    address;
    unsigned char    vr;
    unsigned char    vs;
    unsigned char    nr;
    unsigned char    ns;
    char             inbuf[HDLC_MAX_BUFFERS][HDLC_MAX_INFO_SIZE];
    unsigned char    in_length[HDLC_MAX_BUFFERS];
    unsigned char    current_inbuf;
    unsigned char    next_inbuf;
    char             outbuf[HDLC_MAX_BUFFERS][HDLC_MAX_INFO_SIZE];
    unsigned char    out_length[HDLC_MAX_BUFFERS];
    unsigned char    current_outbuf;
    unsigned char    next_outbuf;
    char             holdbuf[HDLC_MAX_INFO_SIZE];
    unsigned char    hold_length;
    unsigned char    xid_holdoff;
} HDLCSECONDARY_S;
```

```

#define HDLC_HEADER_SIZE
#define HDLC_MIN_FRAME_SIZE      4
#define HDLC_MAX_FRAME_SIZE      (HDLC_MAX_INFO_SIZE + HDLC_MIN_FRAME_SIZE)

// Maximum for XID holdoff random number
#define HDLC_XID_HOLDOFF_MODULUS 32

// Address for broadcast packets
#define HDLC_BROADCAST_ADDRESS    0xff

// Generates P or F bit mask
#define HDLC_PF_BIT(p) (char)((p) ? 0x10 : 0x00)

// Generates receive sequence bit mask
#define HDLC_NS_BITS(ns) (char)(((ns) & 0x07) << 1)

// Generates send sequence bit mask
#define HDLC_NR_BITS(nr) (char)(((nr) & 0x07) << 5)

// Extracts P or F bit
#define HDLC_GET_PF(c) (((c) & 0x10) >> 4)

// Extracts receive sequence number
#define HDLC_GET_NR(c) (char)(((c) & 0xE0) >> 5)

// Extracts send sequence number
#define HDLC_GET_NS(c) (char)(((c) & 0x0E) >> 1)

// HDLC primary control codes
//
// Enter disconnect mode
#define HDLC_PRI_DISC(p) (char)(0x43 | HDLC_PF_BIT(p))

// Set normal receive mode
#define HDLC_PRI_SNRM(p) (char)(0x83 | HDLC_PF_BIT(p))

// Unnumbered information message
#define HDLC_PRI_UI(p) (char)(0x03 | HDLC_PF_BIT(p))

// Test message
#define HDLC_PRI_TEST(p) (char)(0xE3 | HDLC_PF_BIT(p))

```

```

// Exchange ID
#define HDLC_PRI_XID(p) (char)(0xAF | HDLC_PF_BIT(p))

// Receive Ready
#define HDLC_PRI_RR(p, nr) (char)(0x01 | HDLC_PF_BIT(p) | HDLC_NR_BITS(nr))

// Receive NOT ready
#define HDLC_PRI_RNR(p, nr) (char)(0x05 | HDLC_PF_BIT(p) | HDLC_NR_BITS(nr))

// Numbered info message
#define HDLC_PRI_I(p, nr, ns)(char)(0x00 | HDLC_PF_BIT(p) | HDLC_NR_BITS(nr) | HDLC_NS_BITS(ns))

// HDLC secondary control codes
// HDLC secondary device in disconnect mode
#define HDLC_SEC_DM(p) (char)(0x0F | HDLC_PF_BIT(p))

// Unnumbered message acknowledgement
#define HDLC_SEC_UA(p) (char)(0x63 | HDLC_PF_BIT(p))

// Frame reject mode response
#define HDLC_SEC_FRMR(p) (char)(0x87 | HDLC_PF_BIT(p))

// Test message
#define HDLC_SEC_TEST(p) (char)(0xE3 | HDLC_PF_BIT(p))

// Exchange ID
#define HDLC_SEC_XID(p) (char)(0xAF | HDLC_PF_BIT(p))

// Received and ready
#define HDLC_SEC_RR(p, nr) (char)(0x01 | HDLC_PF_BIT(p) | HDLC_NR_BITS(nr))

// Received and NOT ready
#define HDLC_SEC_RNR(p, nr) (char)(0x05 | HDLC_PF_BIT(p) | HDLC_NR_BITS(nr))

// Numbered info message
#define HDLC_SEC_I(p, nr, ns)(char)(0x00 | HDLC_PF_BIT(p) | HDLC_NR_BITS(nr) | HDLC_NS_BITS(ns))

```

```

// HDLC secondary modes
typedef enum
{
    HDLC_STATION_CONNECT_MODE,
    HDLC_NORMAL_RESPONSE_MODE
} SECONDARY_MODES;

// HDLC secondary process results
typedef enum
{
    HDLC_PROCESS_COMPLETED,    // Command was processed
    HDLC_FRAME_FCS_ERROR,      // Frame checksum error
    HDLC_FRAME_SHORT,          // Frame too short
    HDLC_FRAME_LONG,           // Frame too long
    HDLC_FRAME_IGNORED,        // Incorrect mode to receive the frame
    HDLC_WRONG_ADDRESS,        // Not addressed to this secondary
    HDLC_NO_RESPONSE           // No response needed
} SECONDARY_RESULTS;

// Public function prototypes
char hdlc_process_frame(void* p_indata, unsigned char indata_length, void* p_outdata, unsigned char* p_outdata_length);
void hdlc_init(unsigned char secondary_id);
void hdlc_timeout(void);
char hdlc_send_buffer_empty(void);
char hdlc_read_buffer_empty(void);
char hdlc_send_buffer_full(void);
char hdlc_read_buffer_full(void);
unsigned char hdlc_send_data(void* p_data, unsigned char data_length);
unsigned char hdlc_read_data(void* p_data);

#endif // HDLC_H

```