```
/* -*- c++ -*- */
/*
 * Copyright 2015 <F. Stockmayer>.
 */

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <gnuradio/io_signature.h>
#include "crc8_deframer_impl.h"
#include <stdio.h>

#define POLYNOM 0x0700
#define MAX 100000

#define NOSYNC 0
#define PRESYNC 1
#define SYNC 2

namespace gr {
  namespace mymod {

    crc8_deframer::sptr
    crc8_deframer::make()
    {
      return gnuradio::get_initial_sptr
        (new crc8_deframer_impl());
    }

    /*
     * The private constructor
     */
    crc8_deframer_impl::crc8_deframer_impl()
      : gr::sync_decimator("crc8_deframer",
              gr::io_signature::make(1, 1, sizeof(char)),
              gr::io_signature::make(1, 1, sizeof(char)),2)
    {
        init_crc_tab();
        init_dataCorrection_tab();
        init_crcCorrection_tab();
        d_pos = 0;
        d_cnt = 0;
        d_sync = NOSYNC;
    }

    /*
     * Our virtual destructor.
     */
    crc8_deframer_impl::~crc8_deframer_impl()
    {
    }

    int
    crc8_deframer_impl::work(int noutput_items,
                         gr_vector_const_void_star &input_items,
                         gr_vector_void_star &output_items)
    {
        const char *in = (const char *) input_items[0];
        char *out = (char *) output_items[0];

        // Do <+signal processing+>
        for(int i=0; i < noutput_items; i++ ){

          r.d_reg.byte0 = r.d_reg.byte2;
          r.d_reg.byte1 = r.d_reg.byte3;
          r.d_reg.byte2 = in[2*i];
          r.d_reg.byte3 = in[2*i+1];
```

```
            out[i] = get_data(r.d_bits);
        }

        // Tell runtime system how many output items we produced.
        return noutput_items;
    }

    void
    crc8_deframer_impl::init_crc_tab(void){
        int i, j, k;
        unsigned short crc;
        //Calculate CRC8 for all possible 8-Bit Data
        for (i=0; i<256; i++){
                crc = (unsigned short) i;
                // Process 8 Bit Data plus 8 zero filled crc-bits
                for (j=0; j<16; j++) {
                        if ( crc & 0x8000 ) crc = ( crc << 1 ) ^ POLYNOM;
                        else crc = crc << 1;
                }
                crc_tab[i] = (crc >> 8) & 0xff;
        }
    }

   void
   crc8_deframer_impl::init_dataCorrection_tab(void){
        int i;
        for (i=0; i<256; i++){
                dataCorrection_tab[i] = 0;
        }
        dataCorrection_tab[0x07] = 0x01;
        dataCorrection_tab[0x0e] = 0x02;
        dataCorrection_tab[0x1c] = 0x04;
        dataCorrection_tab[0x38] = 0x08;
        dataCorrection_tab[0x70] = 0x10;
        dataCorrection_tab[0xe0] = 0x20;
        dataCorrection_tab[0xc7] = 0x40;
        dataCorrection_tab[0x89] = 0x80;


}

   void
   crc8_deframer_impl::init_crcCorrection_tab(void){
        int i;
        for (i=0; i<256; i++){
                crcCorrection_tab[i] = 0;
        }
        crcCorrection_tab[0x01] = 0x01;
        crcCorrection_tab[0x02] = 0x02;
        crcCorrection_tab[0x04] = 0x04;
        crcCorrection_tab[0x08] = 0x08;
        crcCorrection_tab[0x10] = 0x10;
        crcCorrection_tab[0x20] = 0x20;
        crcCorrection_tab[0x40] = 0x40;
        crcCorrection_tab[0x80] = 0x80;
   }

   char
   crc8_deframer_impl::get_data(uint32_t rx_bits)
    {
        uint32_t bits;
        unsigned char data, mask;
        unsigned char crc_rx;
        unsigned char crc_ok = 0;

        // Shift incoming bits to the correct position
        crc_rx = rx_bits >> (d_pos);
        data = rx_bits >> (d_pos+8);
        syndrom = crc_rx ^ crc_tab[data];
```

```
        // Error Correction
        mask = dataCorrection_tab[syndrom];
        data = data ^ mask;
        mask = crcCorrection_tab[syndrom];
        crc_rx = crc_rx ^ mask;
        syndrom = crc_rx ^ crc_tab[data];

        if(syndrom == 0){
                crc_ok = 1;
        }
        else crc_ok = 0;

        // Synchronization - Octett Alignment
        switch(d_sync){

                case NOSYNC:

                        data = 0;
                        d_cnt = 0;
                        if(crc_ok){
                                d_sync = PRESYNC;
                        }
                        else
                                d_pos = (d_pos+1) % 16;
                        break;

                case PRESYNC:

                        data = 0;
                        if((crc_ok) && (d_cnt == 2)) {
                                d_sync = SYNC;
                                d_cnt = 0;
                        }
                        else if(crc_ok) d_cnt ++;
                        else{
                                d_sync = NOSYNC;
                                d_cnt = 0;
                        }
                        break;

                case SYNC:

                        if(crc_ok) d_cnt = 0;
                        else if(d_cnt < 10) d_cnt ++;
                        else{
                                d_sync = NOSYNC;
                                d_cnt = 0;
                        }
                        break;
        }

        return (data);
    }

  } /* namespace mymod */
} /* namespace gr */
```