Communication protocol

For interaction between the PC and QR a communication protocol is required. Reliability is the key design aspect, although bandwidth limitations should not be overlooked. Additionally, since this protocol needs to execute on both PC and QR, care should be taken that it has limited run-time complexity.

Packet description

Communication occurs via packets of data. Such a packet has a one-byte head and a one-byte tail. Zero or more body bytes may be present in-between. This results in a minimum packet length of two bytes, but no theoretical maximum. Distinction between the three types is possible due to the leading flag bits. A leading `0`, indicates a body byte, a leading `10` indicates a head byte, and a leading `11` indicates the tail byte.

Reliability against data corruption is implemented through a 6-bit checksum in the tail. This checksum is calculated over the entire packet save the reserved checksum space, which is initialized to zero. The CRC algorithm is used, with $x^6 + x + 1$ as its polynomial. Since calculation is expensive a 256-byte lookup-table reduces run-time.

Protection against data loss is inherent in the head-body-tail system, as any type of missing byte is detectable. Though this will result in a discarded packet, synchronicity is not lost and the following packets may be properly received. Additionally, the 6-bit type identifier in the head contains information on the expected body length.

Figure 1: Packet description

Packet							
Head			Body		Tail		
8 bits			$n * 8$ bits		8 bits		
`1`	`0`	6-bit type	`0`	7-bit data	`1`	`1`	6-bit checksum

Limitations

There is currently no way for the sender to know that one of its packages was discarded on the receiving end. A potential solution is the use of acknowledgements, though those would benefit greatly from either sequence numbers or blocking channels.

Secondly, it is only possible to fill a body byte with 7-bits worth of data. As such, if a data structure, or a complete byte would need to be sent over, two translating functions would be required to convert the raw 8-bit format into the required 7-bit format.

There are various limited buffers in use, resulting in maximum sizes. The maximum size for a packet is 32 bytes, dicated by the `FRAME_BUFFER_SIZE`.

Currently `serial_write` is a blocking operation for the PC side. This can easily be changed if necessary.

In several places printfs fulfill the function of the future LOG unit. A search for `//TODO: Log` through all files should yield all of them.

Usage

The protocol requires the following files:

comm.h Include this to make the protocol available. Also contains the packet type enum.

comm.c Protocol code.

checksum.h Header for the CRC-algorithm implementation, including polynomial constants.

checksum.c Checksum calculation and verification code.

serial.h General header for platform-specific RS232 code.

serial.c Linux-specific implementation of serial.h. *Supply this for PC.*

x32_serial.c X32-specific implementation of serial.h. *Supply this for QR.*

The protocol is universal so will work on either on either the PC or QR side. It requires both initialisation and uninitalisation using `comm_init()` and `comm_uninit()` respectively. A return value of `-1` indicates an error. To send and receive one uses the `send_data()` and `recv_data()` functions.

Example

```
#include "comm.h"

void main(void) {
        //Declare variables
        unsigned char axes[4];
        comm_type type;
        unsigned char* data;
        int len;
        int I;

        //Initialise communication
        if (0 != comm_init())
                return;

        //Send keypress of Q
        if (0 != send_data(KEY_Q, 0, 0))
        return;

    //Create dummy axes values
    axes[0] = 0x05; axes[1] = 0x17;
    axes[2] = 0x42; axes[3] = 0x00;
    //Send
    if (0 != send_data(AXES, axes, 4))
        return;

    //Wait until data can be received
    while (0 == recv_data(&type, &data, &len)) {
        //Data was received
        printf("Type:\t%i\nData:\t", type);
        for (I=0; I<len; I++)
            printf("%.2X ", data[I]);
        printf("\n");

        //Free buffer memory again (IMPORTANT!)
        free(data);
    }

    //Uninitialise
    comm_uninit();
}
```