The OpenEEG packet formats P2 and P3

The OpenEEG firmware supports two standard Packet transmission Protocols: P2 and P3. For the MDBA course, we will focus on the P2 protocol, which is easier to understand and to decode. Nonetheless both, P2 and P3, will be introduced here:

Firmware Protocol P2	Firmware Protocol P3
Byte 1: Sync Value 0xa5 Byte 2: Sync Value 0x5a Byte 3: Version Byte 4: Frame Number Byte 5: Channel 1 Low Byte Byte 6: Channel 1 High Byte Byte 7: Channel 2 Low Byte Byte 8: Channel 2 High Byte Byte 9: Channel 3 Low Byte Byte 10: Channel 3 High Byte Byte 11: Channel 4 Low Byte Byte 12: Channel 4 High Byte Byte 13: Channel 5 Low Byte Byte 14: Channel 5 High Byte Byte 15: Channel 6 Low Byte Byte 16: Channel 6 High Byte Byte 17: Button States (b1-b4)	0ppppppx packet header 0xxxxxxx 0aaaaaaa channel 0 LSB 0bbbbbbb channel 1 LSB 0aaa-bbb channel 0 and 1 MSB 0ccccccc channel 2 LSB 0ddddddd channel 3 LSB 0ccc-ddd channel 2 and 3 MSB 0eeeeeee channel 4 LSB 0fffffff channel 5 LSB 1eee-fff channel 4 and 5 MSB 1 and 0 = sync bits. p = 6-bit packet counter x = auxilary channel byte a-f = 10-bit samples from ADC channels 0 - 5 - = unused, must be zero

Note that the auxiliary channel of the P3-protocol can be used to transmit bytes sequenced over a flexible number of packets: There are 8 auxillary channels that are transmitted in sequence, the 3 least significant bits of the packet counter determine what channel is transmitted in the current packet:

Possible Aux Channel Allocations:

```
0: Zero-terminated ID-string (ASCII encoded),
   ID-string is currently "mEEGv1.0".
1: free
2: free
3: free
4: Port D status bits
5: free
6: free
7: free
```

Beside P2 and P3, some other proprietary packet formats have emerged during the evolution of the openEEG project. P21 by Jarek Foltynski allows bidirectional communication, P21 was further modified and extended by Reiner Münch an is in use in the MonolithEEG hardware. Moritz v. Buttlar wrote a protocol externsion for trigger snchronisation and ERP averaging. Have a look at the openEEG firmware and documentatiotn for details

In the following, two packet parser implementations will be shown, one for P2 and one for P3.

P2 Packet Parser :

```
void parse_byte_P2(unsigned char actbyte)
{
 switch (PACKET.readstate)
 {
    case 0: if (actbyte==165) PACKET.readstate++;
      break;

    case 1: if (actbyte==90)  PACKET.readstate++;
           else PACKET.readstate=0;
      break;

    case 2: PACKET.readstate++;
      break;

     case 3: PACKET.number = actbyte;
             PACKET.extract_pos=0;PACKET.readstate++;
      break;

    case 4:
       if (PACKET.extract_pos < 12)
       {   if ((PACKET.extract_pos & 1) == 0)
           PACKET.buffer[PACKET.extract_pos>>1]=actbyte*256;
              else PACKET.buffer[PACKET.extract_pos>>1]+=actbyte;
        PACKET.extract_pos++;
       }
       else
       {
         PACKET.switches= actbyte;
         PACKET.readstate=0;
         process_packets();
       }
       break;
    default: PACKET.readstate=0;
  }
}
```

P3 Packet Parser :

```
void parse_byte_P3(unsigned char actbyte)
{
   int i, j;
   int sync;


  switch (PACKET.readstate)
 {
    case 0: if (actbyte & 0x80) {PACKET.extract_pos=0;PACKET.readstate++;}
      break;

    case 1:
      PACKET.buffer[PACKET.extract_pos]=actbyte;
      PACKET.extract_pos++;
      if (PACKET.extract_pos==11)
      {
        PACKET.number= (PACKET.buffer[0] >> 1) & 0x3f;
        PACKET.aux = (PACKET.buffer[0] << 7) & 0x80
                     | PACKET.buffer[1] & 0x7f;
        if ((PACKET.number & 7)==4) PACKET.switches = PACKET.aux;
        sync = (PACKET.buffer[0] >> 6) & 2 | (PACKET.buffer[1] >> 7) & 1;

        for (i = 0, j = 2; i < (6 >> 1); i++, j += 3)
        {
            // Decode and store even-channel sample
           PACKET.buffer[i << 1] = ((unsigned short) PACKET.buffer[j])
           & 0x7f | (((unsigned short) PACKET.buffer[j+2]) << 3) & 0x380;
           // Decode and store odd-channel sample
           PACKET.buffer[(i << 1) + 1] =
              ((unsigned short) PACKET.buffer[j+1]) & 0x7f
              | (((unsigned short) PACKET.buffer[j+2]) << 7) & 0x380;

          // Decode and store the sync bits
         sync = (sync << 3) | (int) ((PACKET.buffer[j] >> 5) & 4
          | (PACKET.buffer[j+1] >> 6) & 2
          | (PACKET.buffer[j+2] >> 7) & 1);
      }

      if (sync==1) // The sync marker is last, so sync should be == 1
      {
        PACKET.extract_pos=0;
        process_packets();
      } else PACKET.readstate=0;
     }
     break;

    default: PACKET.readstate=0;
  }
}
```