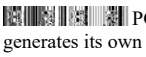
autopilot: Do it yourself UAV

PCM vs PPM

The PPM radio protocol is a very simple serialization of the servo PWM commands, separated by a 10 ms or longer syncronization pulse. In order to decode it, simply wait for the sync pulse then clock the times between the falling edges until the next sync pulse. Code to do this is in onboard/rev2/ppm.h.

However, PPM fails very badly due to local RF noise such as spark plugs or turbine ECUs. These can introduce spikes that cause servo jitter or even erroneous servo commands. At long ranges the analog pulse widths are less accurate as well, leading to jitter. The PPM radio receivers typically have no onboard computers and rely totally on the transmitter to generate the servo pulses. If you lose the transmitter, the servos stop holding their positions. For all these reasons, a better protocol is required.



PCM replaces the pulse widths with a serial bit stream that encodes the servo commands as a binary value. An onboard microcontroller reads the bit stream and generates its own servo commands from it. This MCU can hold the servos in position if the transmitter is briefly out of contact with the receiver and even set the servos into a preprogrammed "fail safe" position if the transmitter is not heard from again after some timeout period.

PCM internals

The actual encodings are considered trade secrets of each manufacturer and they are not compatible across different brands. We have reverse engineered the Futaba PCM1024 protocol and present our analysis here. This was done in a "clean room" fashion using a Futaba 8U transmitter and a modified HiTec RCD3500 receiver to extract the bit stream. The receiver is actually a PPM unit, but the RF front end is the same as the PCM model. Screen shots are of the Tek TDS2012 scope used to analyze the bit stream.

Frame protocol

The PCM frame consists of four fields, each 28 ms long. Each field starts with a 2.7 ms sync pulse of either high or low polarity, follwed either six or eight bits of frame ID, then by four data packets of 40 bits each. Each of these data packets consists of four 10 bit MSB words that are encoded with a 6B10B encoding, to produce 3 bytes of actual data per packet, 12 bytes per field and 48 bytes per frame. From now on, we will assume that all data has been converted to the natural representation.

On the wire, the frame is transmitted like this (with sync pulses between each of the fields):

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>22</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>32</td><td>32</td><td>33</td><td>34</td><td>35</td><td>36</td><td>37</td><td>38</td><td>39</td><td>40</td><td>42</td><td>42</td><td>43</td><td>44</td><td>45</td><td>46</td><td>47</td><td>48</td><td>49</td><td>50</td><td>52</td><td>52</td><td>53</td><td>54</td><td>55</td></tr>
<tr><td colspan="56">PCM Frame</td></tr>
<tr><td>Sync</td><td colspan="13">Field 1</td><td>Sync</td><td colspan="13">Field 2</td><td>Sync</td><td colspan="13">Field 3</td><td>Sync</td><td colspan="13">Field 4</td></tr>
<tr><td>LOW</td><td>1A</td><td>1B</td><td>1C</td><td>1D</td><td>LOW</td><td>2A</td><td>2B</td><td>2C</td><td>2D</td><td>HIGH</td><td>3A</td><td>3B</td><td>3C</td><td>3D</td><td>HIGH</td><td>4A</td><td>4B</td><td>4C</td><td>4D</td></tr>
</table>

The bit clock is 150 us. There does not appear to be any syncronization mechanism other than the accuracy of the onboard clock. Bits are sampled roughly 25 us after a transition and there never appears to be a single bit alone (minimum pulse width is 200 us). At first I thought the bit clock was 300 us, but Coert L. showed me transitions that had to be sampled at the 150 us rate.

I've numbered the four fields starting with the first low sync pulse. This is because the fail safe data are transmitted twice, starting with the low sync pulse. However, since the data are transmitted twice, it will overlap an entire frame regardless of which ordering you use. There are two low sync pulse fields (1 and 2), followed by two high sync pulse fields (3 and 4) which are nominally logical inversions of the first two fields.

Data packets

The data packet format consists of 2 bits of type selectors, 4 bits of position delta, 10 bits of absolute position and 8 bits of checksum. Bit zero is transmitted first:

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td></tr>
<tr><td>Selector</td><td>Delta</td><td colspan="10">Position</td><td colspan="10">Checksum</td></tr>
</table>

I have not fully analyzed the type selection bits yet, but they appear to select between absolute data for servo Channels 7, 8, 9 and 10 in some fashion, as well as indicating fail safe frames. The delta value is treated as an offset from the previous absoluate position frame for this channel, with a value of 8 meaning no change. The position is treated as a 10 bit MSB value, hence the name PCM1024. The checksum appears to be a sum of the position and other bits; I have not fully analyzed it yet.

Checksum computation

To be written

Channel assignment

As mentioned before, each field has four data packets that we're calling A, B, C and D. The first packet in field 1 is 1A and so on. The position and delta channels for each normal data packet appear to be:

```
1A:
  Select=0 ???
  Select=1 ???
  Select=2 Channel 1 absolute, Channel 2 delta
  Select=3 ???
1B:
  Select=0 Channel 3 absolute, Channel 4 delta
  Select=1 ???
  Select=2 ???
  Select=3 ???
```

```
1C:
  Select=0 ???
  Select=1 ???
  Select=2 Channel 5 absolute, Channel 6 delta
  Select=3 ???
1D:
  Select=0 Channel 7 absolute, Channel 8 delta
  Select=1 ???
  Select=2 ???
  Select=3 ???

2A:
 Select=0 ???
  Select=1 ???
  Select=2 Channel 2 absolute, Channel 1 delta
  Select=3 ???
2B:
  Select=0 Channel 4 absolute, Channel 3 delta
  Select=1 ???
  Select=2 ???
  Select=3 ???
2C:
  Select=0 ???
  Select=1 ???
  Select=2 Channel 6 absolute, Channel 5 delta
  Select=3 ???
2D:
  Select=0 Channel 8 absolute, Channel 7 delta
  Select=1 ???
  Select=2 ???
  Select=3 ???
```

Fail safe frames

To be written.

Appendix 1: 6B10B encoding

Thanks to Coert Langkemper for his help in determining the encoding pattern. Without his aid, this would have taken absolutely forever.

```
        '1111111000'    => 0x00,
        '1111110011'    => 0x01,
        '1111100011'    => 0x02,
        '1111100111'    => 0x03,
        '1111000111'    => 0x04,
        '1111001111'    => 0x05,
        '1110001111'    => 0x06,
        '1110011111'    => 0x07,
        '0011111111'    => 0x08,
        '0001111111'    => 0x09,
        '0000111111'    => 0x0A,
        '1100111111'    => 0x0B,
        '1100011111'    => 0x0C,
        '1100001111'    => 0x0D,
        '1110000111'    => 0x0E,
        '1111000011'    => 0x0F,
        '0011111100'    => 0x10,
        '0011110011'    => 0x11,
        '0011100111'    => 0x12,
        '0011001111'    => 0x13,
        '1111001100'    => 0x14,
        '1110011100'    => 0x15,
        '1100111100'    => 0x16,
        '1100110011'    => 0x17,
        '1111110000'    => 0x18,
        '1111100000'    => 0x19,
        '1110000011'    => 0x1A,
        '1100000111'    => 0x1B,
        '1100011100'    => 0x1C,
        '1110011000'    => 0x1D,
        '1110001100'    => 0x1E,
        '1100111000'    => 0x1F,
        '0011000111'    => 0x20,
        '0001110011'    => 0x21,
        '0001100111'    => 0x22,
        '0011100011'    => 0x23,
        '0011111000'    => 0x24,
        '0001111100'    => 0x25,
        '0000011111'    => 0x26,
        '0000001111'    => 0x27,
        '0011001100'    => 0x28,
        '0011000011'    => 0x29,
        '0001100011'    => 0x2A,
        '0000110011'    => 0x2B,
        '1100110000'    => 0x2C,
        '1100011000'    => 0x2D,
        '1100001100'    => 0x2E,
        '1100000011'    => 0x2F,
        '0000111100'    => 0x30,
        '0001111000'    => 0x31,
        '0011110000'    => 0x32,
        '0011100000'    => 0x33,
        '0011000000'    => 0x34,
        '1111000000'    => 0x35,
```

```
    '1110000000'    => 0x36,
    '1100000000'    => 0x37,
    '0001100000'    => 0x38,
    '0001110000'    => 0x39,
    '0000110000'    => 0x3A,
    '0000111000'    => 0x3B,
    '0000011000'    => 0x3C,
    '0000011100'    => 0x3D,
    '0000001100'    => 0x3E,
    '0000000111'    => 0x3F,
```

$Id: pcm.html,v 1.3 2003/02/03 19:42:41 tramm Exp $