Co Processor Specifications

Summary

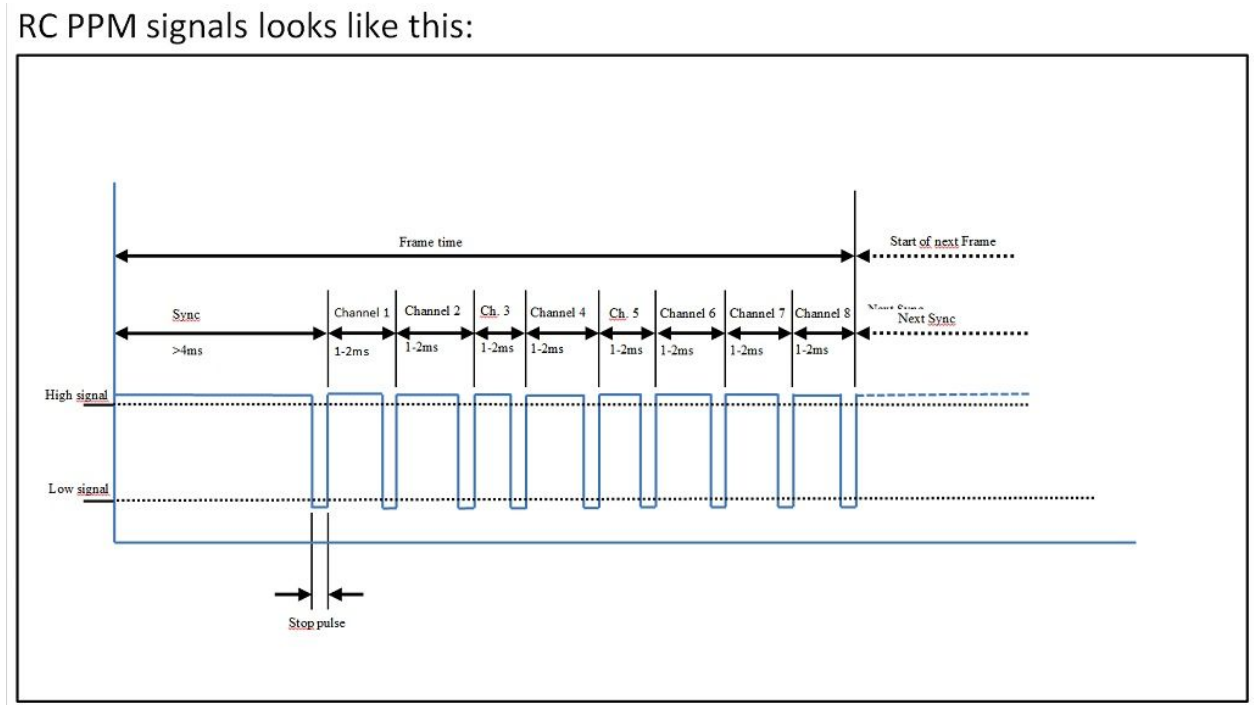
The goal of the coprocessor is to connect the onboard computer to the vehicle. It will allow the onboard computer to talk to the vehicle's radio controller, steering, and throttle. It will do this by decoding PPM(Pulse Position Modulation) signals and encoding PWM signal(Pulse Width Modulation) and communicating with onboard computer over serial.

PPM Decoder

Summary

The job of the PPM decoder is to decode the PPM signal from the RC receiver onboard the vehicle.

PPM

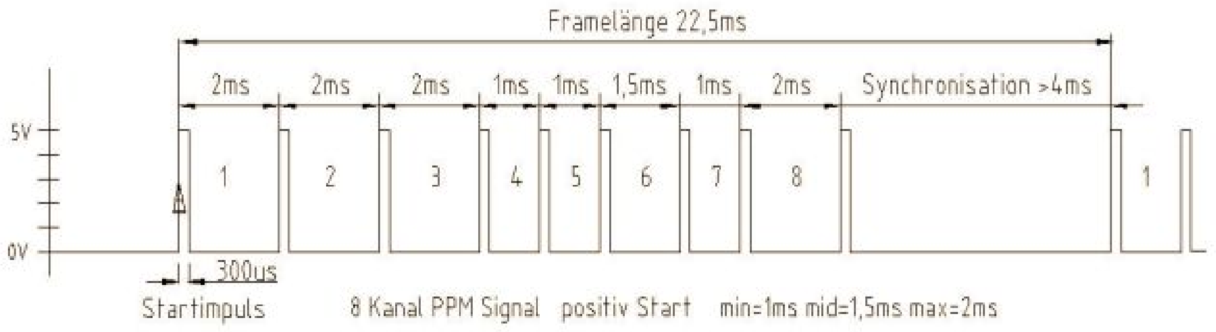


PPM stands for Pulse Position Modulation and can encode multiple channels of information on a single wire(It also used interchangeably with CPPM). A channel contains a value from 500-2000. Standard PPM encodes 8 channels. PPM is a frame based one way communication. A frame encodes all channels and starting/ending pulses. Frames usually come(last) every

20ms. Channels are encoded into frames using pulse length. The length of the pulse in microseconds directly corresponds to the value for that channel(i.e. 1000us pulse -> 1000). Channel pulses are sent sequentially. The first pulse corresponds to the first channel and so on. There are gaps,called stop pulses, between channels pulses in order to separate channels. PPM can be encoded active high or active low. Active high means channels are encoded over high pulses and gaps are on low pulses. Active low inverts this. This characteristic is dependent on the PPM receiver. Another thing that is dependent on the manufacturer is frame width and starting and end pulses.

FR Sky PPM

The PPM receiver our team is using the FR Sky DR4-II. **It is a 4 channel receiver in PWM mode but 8 channel in PPM mode.** A jumper must be placed between 2 pins in order to put it into PPM mode. See image on Fr sky PPM below.



Notes:

- This signal is active low
- stop pulse is actually incorporated into the total time for the channel
- Frame length 22.5ms

Arduino Implementation

The best way to decode PPM in a non blocking way is to use hardware pin interrupts. These interrupts are triggered on rising/falling edges and call an ISR(interrupt service routine) function. **The end goal is to have an int array that contains all 8 channels values and is updated every 20ms in a non blocking manner.** Even though we'll only need 2~4 channels, it is easiest to decode and store all 8. There are many examples of people who have made arduino PPM decoders so we should not have to write one from scratch. Listed below are examples:

1. https://github.com/claymation/CPPM
2. https://github.com/jantje/ArduinoLibraries/tree/master/RCLib

Note: Some implementations use hardware timers which may be used by other pins for PWM generation. So we must be careful that we properly allocate our hardware timers so they don't overlap.

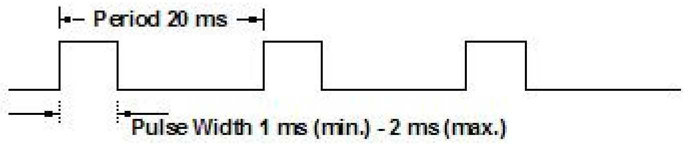
PWM Encoder

Summary

The goal of the PWM encoder is to encode PWM signals for the electronic speed controller(ESC) and steering servo.

PWM

PWM stands for Pulse Width Modulation and it encode a single channel of data on a single line. A channel can encode a data value from 1000 to 2000. Like PPM, PWM is frame based but much simpler. PWM frames occur every 20 ms. The first part of the frame is a high pulse of length 1000 microseconds to 2000 microseconds. Like PPM, microseconds correspond directly with channel value. The remaining time of the frame is a low pulse 18000 - 19000 us seconds long. Then the next frame occurs. There are no start or stop pulses since it is a single channel.



Servo vs ESC

Both the servo and ESC operate using 1000-2000 microsecond PWM signals. For *most* servos 1500us is the center point(90 degrees), 1000 us means left(0 degrees) and 2000 us means right(180 degrees). For the ESC 1500us means neutral, 1000us is full reverse, and 2000us is full forward.

Arduino Implementation

Arduino has built in PWM generation so it only requires a few lines of code to move a servo/ESC. We will use the Servo.h library and writeMicroseconds() generate PWM. Since PWM uses hardware timers, not all pins support it. This means we have to be careful with our pin selection and also make sure it doesn't interfere with the PPM decoder.

Communication

Summary

The co processor will communicate with the onboard computer using usb serial. This allows the onboard to send commands to vehicle while monitoring the user input from the receiver.

Data

The data sent across the serial lines will be bi-directional. The data will packaged in packets and will be defined by a specific set of messages.

- **Heartbeat**: Sent at a constant interval by both devices to verify they are alive. Contains the current mode.
- **Channels_in**: Decoded RC receiver channels sent at a constant interval from coprocessor. Don't need to monitor all 8. So we only send 4
- **Control:** Sent from the onboard computer to control the throttle and steering
- **Set_mode**: Sent from the onboard computer to change the operation mode of the coprocessor
- **Error:** For sending error codes to the onboard computer
- **Debug:** For sending debug messages to the onboard computer

Packet Format

Packets are byte arrays that contain messages. The packet format will be as follows.

Byte 0: start byte
Byte 1: Message id
Byte 2+: payload

The message length and payload will depend on the message id.
Msg_length = start_byte(1) + msg_id(1) + payload_len

Msg_id	**Msg_type**	**Msg_length**	**Field 1**	**Field 2**	**Field 3**	**Field 4**
0x00	Heartbeat	3	Mode - enum(1)			
0x01	Channels_in	10	Ch1 - short(2)	Ch2 - short(2)	Ch3 - short(2)	Ch4 - short(2)
0x02	Control	6	Throttle - short(2)	Steer - short(2)		
0x03	Set_mode	3	Mode - enum(1)			
0x04	Error	5	Error Code - byte(1)	Error data - short(2)		

0x05	Debug	14	Value 1 - long(4)	Value 2 - long(4)	Value 3 - long(4)	

Data Rate

- **Heartbeat**
    - All Mode: 2Hz bidirectional
- **Channels_in**
    - All Modes: 50hz to onboard computer
- **Control**
    - Manual Mode: Not sent
    - Auto Mode: Framerate -> 5~30hz
- **Set_mode**
    - All Modes: Not sent on an interval. Sent when mode change requested

Error codes

Error Code	**Name**	**Description**
0x00	Invalid msg_id	Message ID not supported
0x01	No RC	Lost RC communication
0x02	No Heartbeat	Lost Serial contact
0x03	Invalid value range	Packet contained data with invalid range

Alternative: Have an error bit mask and error extra fields that gets sent with heartbeat. Could be more real time, light weight approach.

Modes of operation

Summary

The coprocessor can run in 3 modes. Manual mode which will act as a passthrough mode for collecting data from human driver, and Auto mode which will accept controls commands from the onboard computer, and Failsafe mode which will shut down the vehicle when communication fails.

Manual Mode

Manual mode is simply a pass through mode that will be used for user control of the vehicle. Manual mode will decode PPM from the receiver and encode it to PWM to the servo and ESC. The decoded PPM will be packaged and sent to onboard computer using the channels_in message. If the PPM signal goes into failsafe state, which means a radio disconnect, the coprocessor will switch to failsafe mode. Switching out of manual mode can be achieved using the set_mode message or changing the value of channel 3 on the RC transmitter.

Auto Mode

Auto mode is computer control mode. Auto mode will decode PPM from the receiver and send it to the onboard computer. It will also receive control messages from the onboard computer and encode it to PWM for the servo and ESC. If the PPM signal goes into failsafe state, which means a radio disconnect, the coprocessor will switch to failsafe mode. Switching out of auto mode can be achieved using the set_mode message or changing the value of channel 3 on the RC transmitter.

Failsafe Mode

Failsafe mode occurs when heartbeat messages timeout, the RC transmitter disconnects, or the user switches into failsafe mode. Failsafe mode puts the ESC in neutral and centers the steering servo. The user or computer can only switch out of failsafe mode if the RC transmitter is detected and heartbeat messages are streaming. If these conditions aren’t met and a mode change is attempted then an error message will be sent. Switching out of auto mode can be attempted using the set_mode message or changing the value of channel 3 on the RC transmitter.