2. Subscriber tutorial

Introduction

In this article we will extend the functionality of the UAVCAN node from the previous tutorial. For pragmatic reasons we will make something useful - a UAVCAN to RCPWM converter.

> **OBSOLETE**
> THE MATERIALS PRESENTED HERE ARE BASED ON **UAVCAN v0** – AN EXPERIMENTAL VERSION OF THE PROTOCOL THAT IS NOW OBSOLETE. TO LEARN ABOUT THE LONG-TERM STABLE REVISION OF THE PROTOCOL, PROCEED TO UAVCAN.ORG.

Goal

Our goal is to make Zubax Babel act as a UAVCAN node and receive RCPWM values via CAN and then convert them to RCPWM signal outputs.

A couple of words about RCPWM. RCPWM is one of the most popular low-level analog interfaces in robotics. Most of the low-cost servos and motor controllers support it (and there are lots and lots of other devices that communicate this way). Without a doubt, it is a very old, slow and limited interface, and today it must be replaced with something better (like UAVCAN). But this is legacy and we have to deal with it. From the physical standpoint, RCPWM (also known as RC PWM) is just a PWM signal with a period of 20 ms where the information is encoded in pulse width which may vary from 1 ms to 2 ms with the neutral value at 1.5 ms. To generate that, we will use timers that are available on Zubax Babel (according to the datasheet where we can also find a detailed pinout specification).

We will use `TIM12_CH1`, `TIM12_CH2`, `TIM3_CH1`, `TIM3_CH2`, `TIM3_CH3`, `TIM3_CH4`, utilizing all available pins.

Implementation

We will base our project on the code from the Basic tutorial and add the missing functions. Firstly we should add support of a new message type to `shouldAcceptTransfer` and its appropriate handler to `onTransferReceived`:

shouldAcceptTransfer

```
static bool shouldAcceptTransfer(const CanardInstance* ins,
                                 uint64_t* out_data_type_signature,
                                 uint16_t data_type_id,
                                 CanardTransferType transfer_type,
                                 uint8_t source_node_id)
{
    if ((transfer_type == CanardTransferTypeRequest) &&
        (data_type_id == UAVCAN_GET_NODE_INFO_DATA_TYPE_ID))
    {
        *out_data_type_signature =
UAVCAN_GET_NODE_INFO_DATA_TYPE_SIGNATURE;
        return true;
    }

    if(data_type_id == UAVCAN_EQUIPMENT_ESC_RAWCOMMAND_ID)
    {
        *out_data_type_signature =
UAVCAN_EQUIPMENT_ESC_RAWCOMMAND_SIGNATURE;
        return true;
    }

    return false;
}
```

onTransferReceived

```
static void onTransferReceived(CanardInstance* ins, CanardRxTransfer*
transfer)
{
    if ((transfer->transfer_type == CanardTransferTypeRequest) &&
    (transfer->data_type_id == UAVCAN_GET_NODE_INFO_DATA_TYPE_ID))
    {
        getNodeInfoHandleCanard(transfer);
    }

    if (transfer->data_type_id == UAVCAN_EQUIPMENT_ESC_RAWCOMMAND_ID)
    {
        rawcmdHandleCanard(transfer);
    }
}
```

We added support of `UAVCAN_EQUIPMENT_ESC_RAWCOMMAND_ID`, message that contains all values needed to generate RCPWM signal from ESC setpoint messages. The UAVCAN GUI Tool starts to broadcast these messages when the ESC Management panel is opened. Note that other messages can also be mapped to RCPWM outputs; for example, the actuator command messages from the namespace `uavcan.equipment.actuator`.

Now it's the handler's turn, where RCPWM values are passed to the MCU timers:

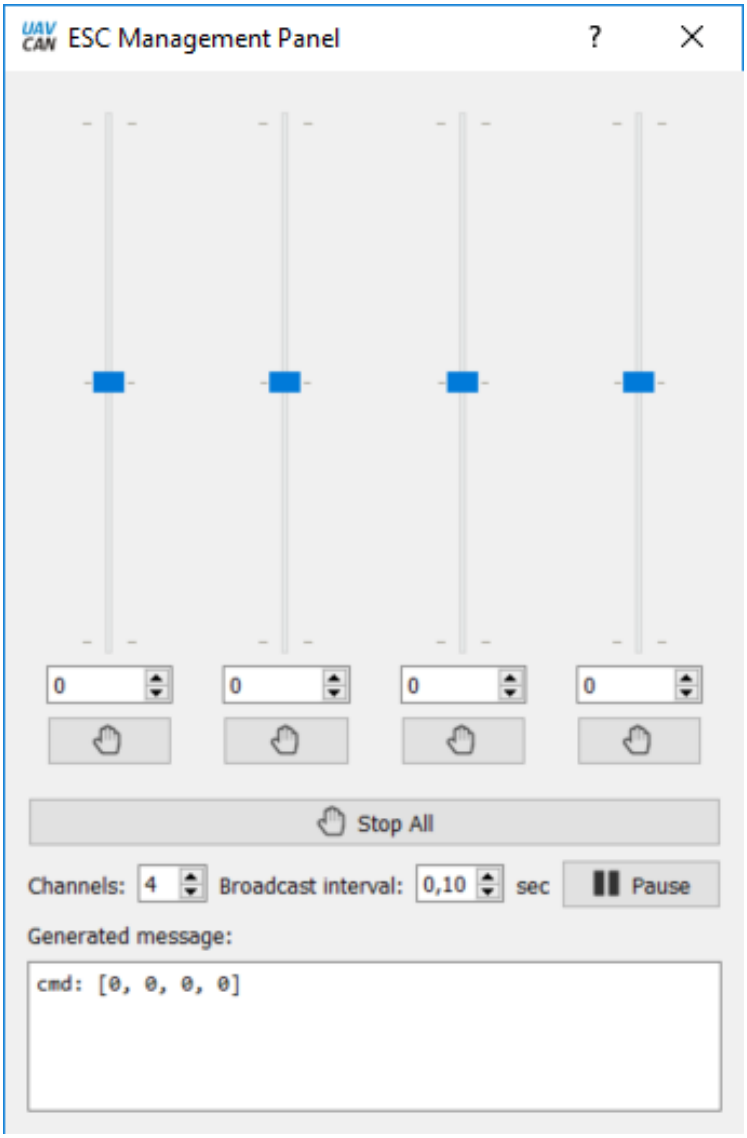
canard_rawcmd_handle

```
static void rawcmdHandleCanard(CanardRxTransfer* transfer)
{
    int16_t ar[6] = {0,0,0,0,0,0};
    int offset = 0;
    for (int i = 0; i<6; i++)
    {
        if (canardDecodeScalar(transfer, offset, 14, true, &ar[i])<14) {
break; }
        offset += 14;
    }
    rcpwmUpdate(ar);
}

const rcpwm_t rcpwm_outputs[] =
{
    {TIM12, 1, "TIM12_CH1_PA4"},
    {TIM12, 2, "TIM12_CH1_PA5"},
    {TIM3,  1, "TIM3_CH1_PA6"},
    {TIM3,  2, "TIM3_CH2_PB0"},
    {TIM3,  3, "TIM3_CH3_PB6"},
    {TIM3,  4, "TIM3_CH4_PB7"}
};

static void rcpwmUpdate(int16_t ar[6])
{
    for (char i = 0; i < 6; i++ )
    {
        uint32_t val = RCPWM_NEUTRAL_uS;
        ar[i] = ar[i] * RCPWM_MAGNITUDE_uS /
```

```
UAVCAN_EQUIPMENT_ESC_RAWCOMMAND_MAX_VALUE;
        val += ar[i];
        TIM_SetCompare(rcpwm_outputs[i].timer,rcpwm_outputs[i].channel,
uS_to_ticks(val));
    }
}

static void TIM_SetCompare(TIM_TypeDef* TIMx, uint8_t ch, uint32_t
Compare)
{
    if (ch == 1)
    {
        TIMx->CCR1 = Compare;
        return;
    }
    if (ch == 2)
    {
        TIMx->CCR2 = Compare;
        return;
    }
    if (ch == 3)
    {
        TIMx->CCR3 = Compare;
        return;
    }
    if (ch == 4)
    {
        TIMx->CCR4 = Compare;
```

```
            return;
        }
    }
```

And that's it. Now its time to open the UAVCAN GUI Tool and go to the ESC Management panel:



Our device supports up to six channels, so increase the number of channels and try moving the sliders.

If you connect an oscilloscope to Babel, you should see something like this:

Or you may connect servos and see them move after the sliders on the ESC panel.