Drone Payload Updates

28 February 2024

Contents

1 Introduction and current hardware

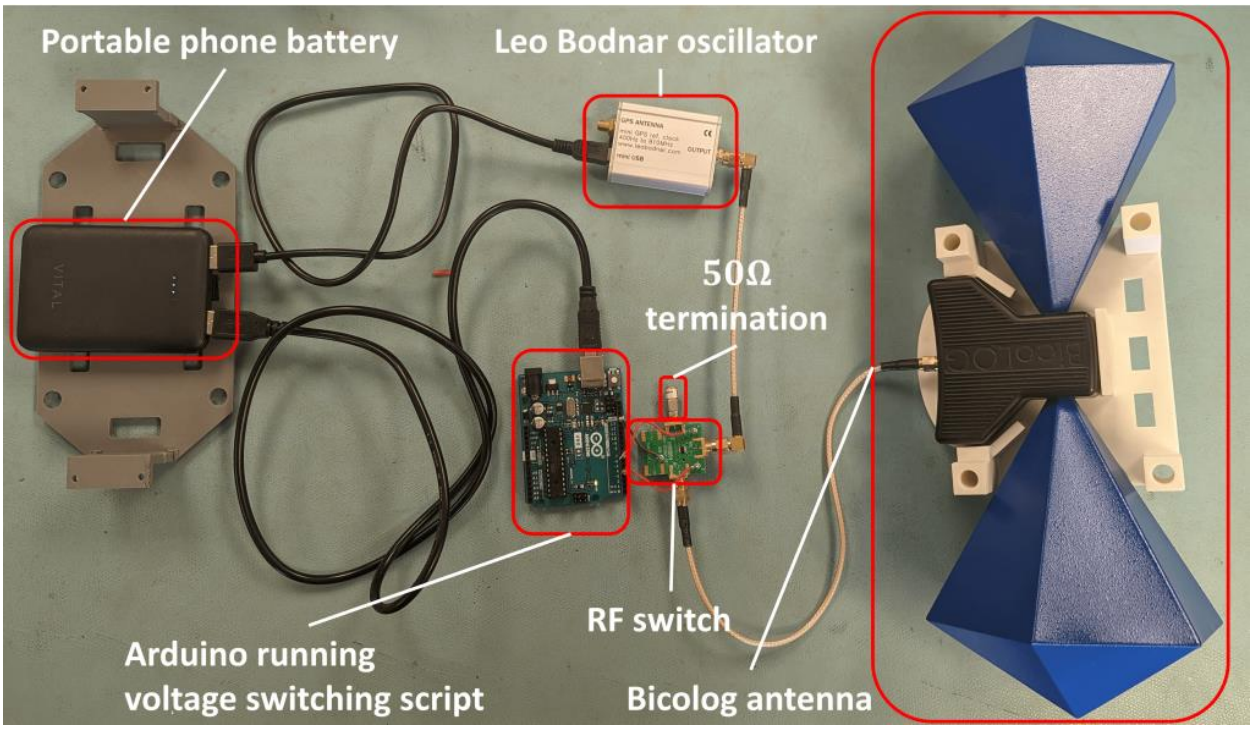
Until now, the PteroSoar drone radio payload has consisted of the following:

Signal source: Leo Bodnar mini precision GPS reference clock[1]

Chopper: Analog Devices HMC545 RF switch eval board[2] controlled by an Arduino Mega[3]

Battery: cellphone battery pack from Amazon with two USB-A outputs

Antenna: Aaronia BicoLog 5070[4]



To switch from a steady-on state to a chopping state, it has been necessary to land the drone and manually change configuration. This is not desired as it needlessly drains battery in the descent to land and climbout back to pattern altitude during the calibration flight at the start of a series of mapping sorties.

The chopping script used on the Arduino has been a simple loop using timeouts to control the GPIO inputs to the HMC545 rf switch. This method of using timeouts is not very accurate, as the Arduino is not perfect at measuring time and the chopper can drift over even short flights. In addition, this setup is needlessly bulky and expensive:

Leo Bodnar cost: CAD $187.75[1]

HMC545 eval board cost: CAD $188.68[2]

Arduino Mega cost: CAD $48.40[3]

2 New Hardware

2.1 Overview

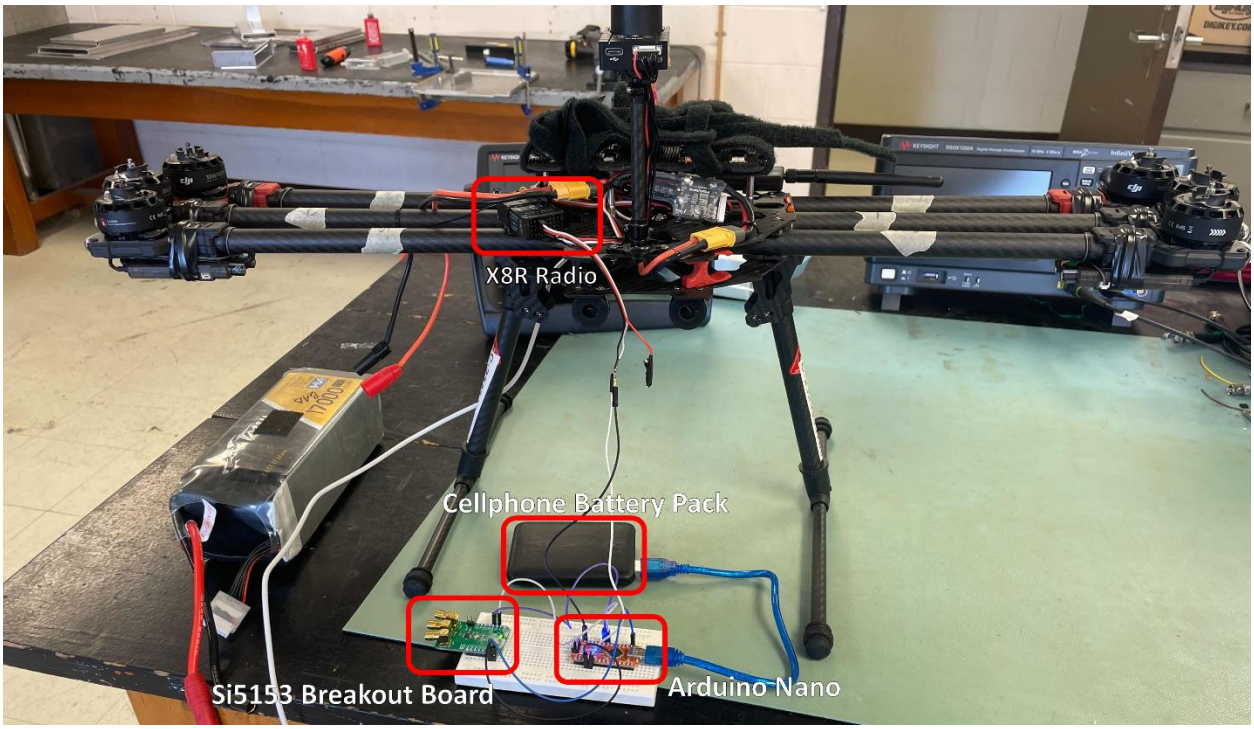
Complexity, bulkiness, weight, and cost can be reduced by using a smaller Arduino and a clock generator controlled by the Arduino GPIO. The chopper can be made more accurate and chopping frequency drift minimized by using the internal timer/counter of the Arduino's processor. Then, the chopping can be implemented by turning on and off the output of the clock generator at a cadence controlled by the timer/counter. There is no need for the RF switch in this configuration, only the Arduino and the clock generator chip. Moreover, by tying the drone's X8R radio into the Arduino GPIO as an input, we can use the same timer/counter to read the PWM signal and implement radio control of the chopper state (steady on, chopping, or steady off).

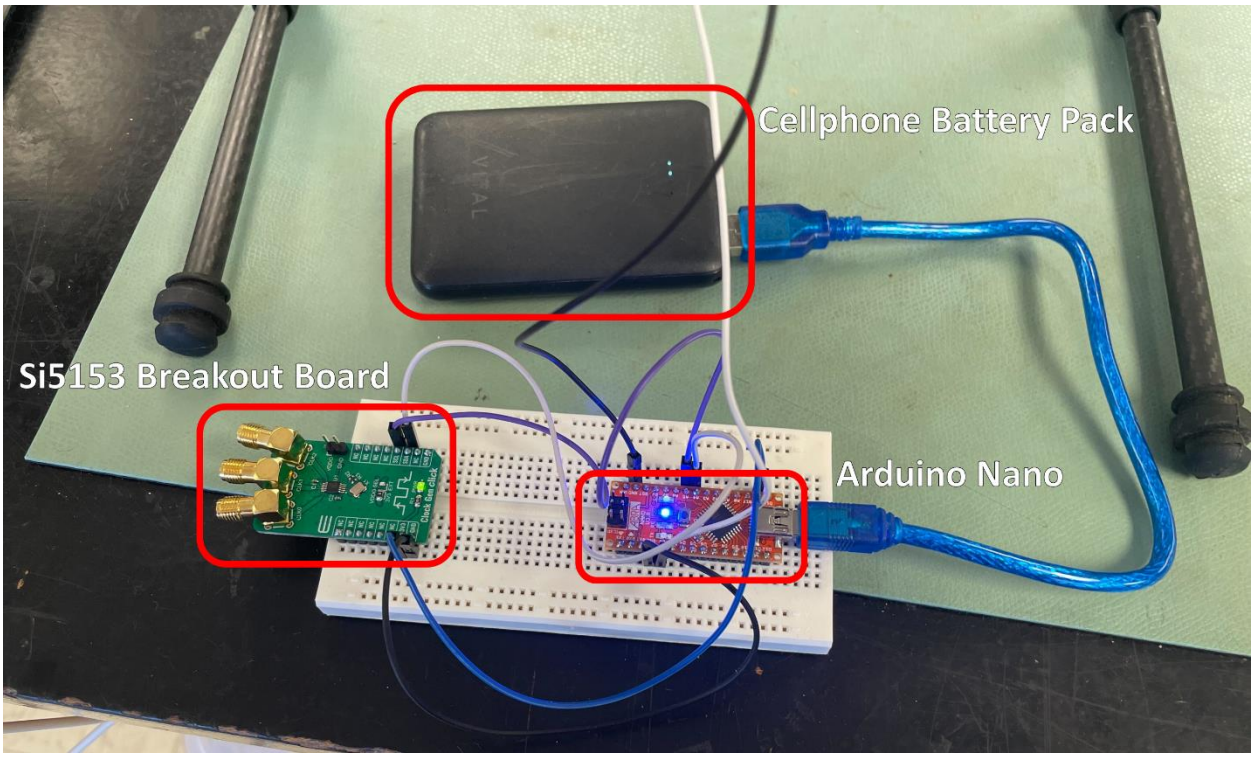
In this configuration we have employed an Arduino (Abra) Nano[5] controlling an Si5351 clock generator chip on a Mikroe-4113 eval board[6]. Abra is a local electronics store and their Abra Nano is functionally the same as a stock Arduino Nano.

Arduino Nano cost: CAD $19.45[5]

Mikroe-4113 Si5351 eval board cost: CAD $31.33[6]

Cost savings: CAD $374.05





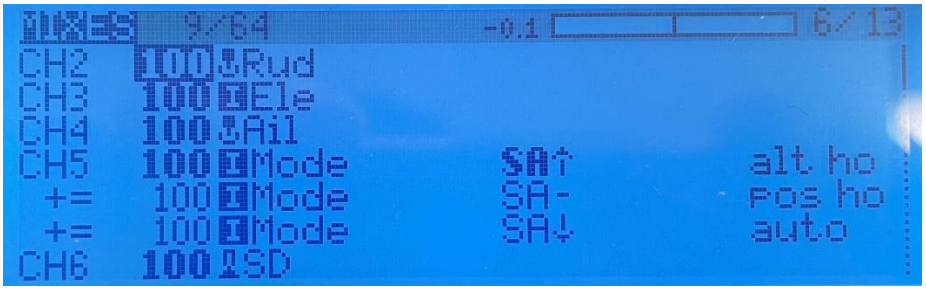
In summation, the internal timer/counter of the Arduino is used for two functions: implementing chopping by toggling the output of the clock generator at a precise rate of 10 Hz; and reading the PWM output of the X8R radio (which corresponds to the position of a switch on the Taranis and sets the mode of the chopper – steady on, chopping, or steady off).

2.2 Wiring

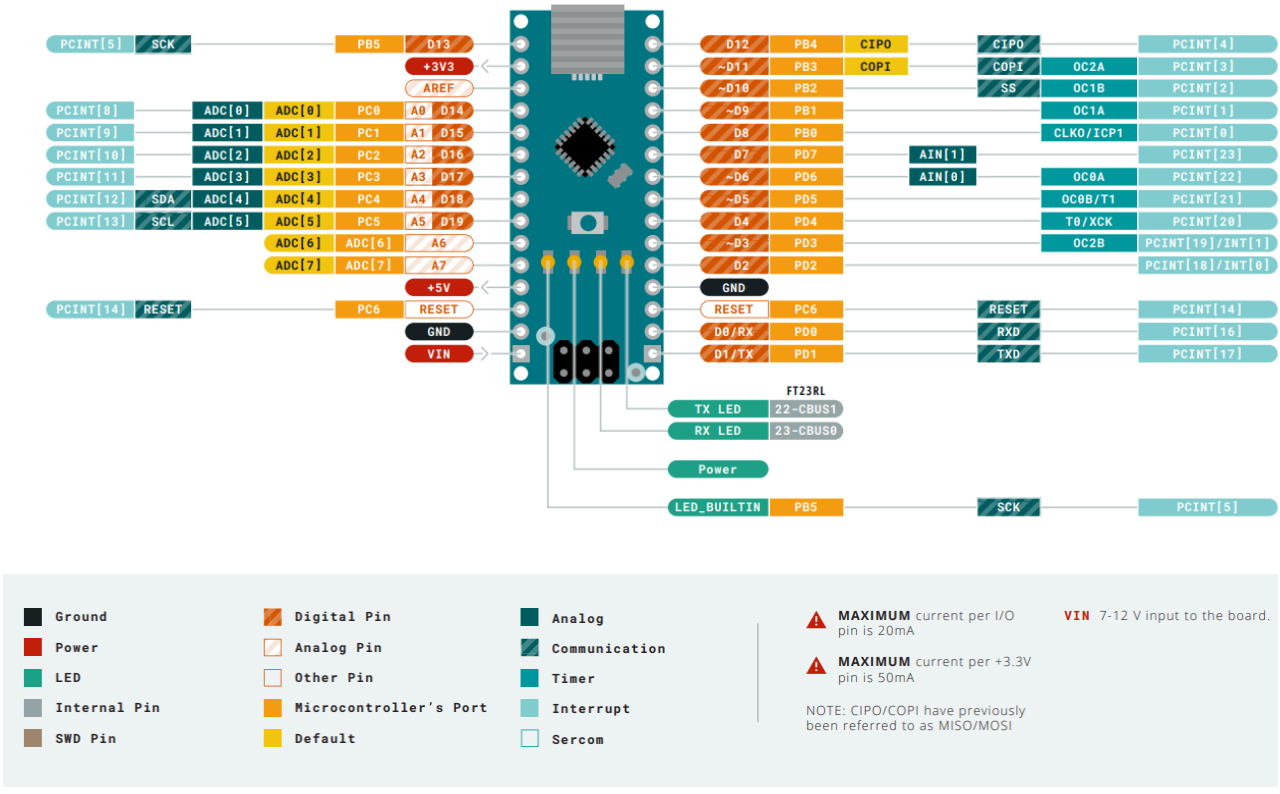
The Si5153 is controlled via I2C and runs off of 3.3 V. The Arduino's FTDI USB chip supplies enough current to power the Si5153. For power the 3V3 and GND pins of the Si5153 eval board are connected directly to the 3.3 V tap off of the Arduino and one of its ground pins. If this proves problematic when connected to load, a 3.3 V regulator will be easy to implement. The Arduino Nano's I2C SDA and SCL pins are pins A4 and A5 by default, and these are wired directly to the SDA and SCL pins of the Si5153 eval board.

The signal and ground pins from output 6 of the X8R radio are connected to pin D8 (the default input capture interrupt pin for the Arduino's timer/counter – more on that in a bit) and a ground pin, respectively. Output 6 is used because we are using channel 6 (Taranis switch D mapped to channel 6 in the Taranis "Mixes" config page). Channels 1-5 are used for drone control, so channels 6-8 are available for other uses such as the chopper.

Taranis "Mixes" config page, note CH6:



Arduino Nano pinout, for reference:



Here we can see the I2C SDA and SCL pins on A4 and A5. We can also see the pins concerning the timer/counter:

Output compare pins OC1A and OC1B on pins D9 and D10

Input capture pin ICP1 on D8

The 3.3 V tap off and ground pins are also easily identified.

3 Arduino code

3.1 Setting up the Si5153 clock generator

The Si5153 has two PLLs and three outputs. To have three different output frequencies, two outputs would need to share one of the PLLs.

Each PLL is set with a multiplier from the default clock speed of 25 MHz. Then each clock output can have two different dividers to set an output frequency.

In integer mode, the two PLL multipliers can have values of 15 to 90. The first clock divider (called multisynth divider) can have values of 4, 6, or 8. The second divider (called additional R divider) can have values of 1, 2, 4, 8, 16, 32, 64, 128. SiLabs recommends integer mode to avoid clock jitter, though fractional settings are also available.

SiLabs puts out some configuration software to help optimize the combinations of PLL multipliers, multisynth dividers, and R dividers for whatever set of output frequencies one desires.

There are two Arduino libraries for the Si5153: one from Adafruit[7], and one from Etherkit[8]. The Adafruit library gives a bit more intuitive low-level setting of the individual PLL multipliers and dividers. However, the Adafruit library doesn't have individual on-off control of the three outputs – either all three are on or all three are off.

The Etherkit library does have the ability to set PLLs, but it is functionally "clunky" and is not designed with that kind of manipulation in mind. Instead, the user provides a frequency, and the library determines what multipliers and dividers are necessary to set the desired frequency. The library cautions that if the desire is to set the PLL multisynths and dividers, then the user must keep careful track of the settings to make sure valid multipliers and dividers are used.

Two advantages of the Etherkit library are the ability to turn on and off each of the three outputs individually, and the ability to set the power levels of the three outputs (drive strength from 2 mA or ~ 3 dBm to 8 mA or ~10 dBm, according to Etherkit). Currently, we are using the Etherkit library but the Adafruit version would be simple to implement.

The Si5153 manipulation in the Arduino code is simple. In the setup function, the chip is initialized, the frequency set, and all 3 outputs disabled. As described later, the output of clock 0 is then toggled continuously for chopping or set to on or off for the non-chopping modes.

3.2 Enabling and setting up the Arduino's timer/counter

3.2.1 Timer/counter overview

The timer/counter is an integral part of the Atmel ATMega328p microcontroller that is the heart of the Arduino. Usage of this chip is well-documented in its data sheet[9]. A timer counter increments at the clock speed of the processor, and in normal mode it overflows at its max value and starts over. We use a 16-bit timer-counter with a max value of 65535. Rather than normal mode, we use it in clear timer on compare mode. In this mode, the timer/counter increments to a preset maximum value, resets, and resumes counting. It can be configured to trigger an interrupt at the maximum value and execute a set of instructions. Prescalers for the clock can also be set, which divide down the clock speed to increment the counter at effectively a slower rate. This is useful if trying to time a longer period than would be possible by incrementing to the max value of 65535 at the Arduino's clock speed of 16 MHz.

The Arduino has three timer/counters: 0 which is 8-bit, 1 which is 16-bit, and 2 which is 8-bit. We use timer/counter 1 as the 16 bits give finer control over the chopping frequency, and also timer/counter 1 is the only one of the three that has an input capture bit to use for PWM measurement. Section 15 of the data sheet[9] describes timer/counter 1.

3.2.2 Setting the timer/counter registers

From the data sheet[9], there are two control registers for timer/counter 1 that need to be set: TCCR1A and TCCR1B. We also need to set register TIMSK1, the interrupt mask register. We use interrupts for both the 10 Hz chopping and the PWM measurement. We will also set the data direction register.

Both registers TCCR1A and TCCR1B contain waveform generation mode bits that must be set, described in the following table.

Table 15-5. Waveform Generation Mode Bit Description(1)

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, phase correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, phase correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, phase correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, phase and frequency correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, phase and frequency correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, phase correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, phase correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Note: 1. The CTC1 and PWM11:0 bit definition names are obsolete. Use the WGM12:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

We want mode 4: clear timer on compare (CTC). It will count to its maximum value which we set in register OCR1A (output compare register). We could also use mode 12, but that uses the input capture register ICR1 for its maximum counter value. We want to use that register in measuring the PWM output of the drone's X8R radio by having the PWM signal trigger an input capture interrupt, so we use mode 4 instead.

3.2.2.1 Setting timer/counter control register TCCR1A

From the datasheet[9], the bits of TCCR1A that we need to set are:

15.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bits WGM11 and WGM10 can be found in table 15-5 above. For mode 4 they are both set to 0. The remaining bits can be found in the following table:

Table 15-2. Compare Output Mode, non-PWM

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on compare match.
1	0	Clear OC1A/OC1B on compare match (set output to low level).
1	1	Set OC1A/OC1B on compare match (set output to high level).

Table 15-3 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to the fast PWM mode.

We will toggle both OC1A and OC1B on compare match (i.e. when the timer/counter reaches its max value), just to have two outputs (on pins D9 and D10) should we ever want that. So, we set COM1A1 and COM1B1 to 0, and COM1A0 and COM1B0 to 1.

Thus, we set TCCR1A to a state of: B01010000.

3.2.2.2 Setting timer/counter control register TCCR1B

From the datasheet[9], the bits of TCCR1B that we need to set are:

15.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit ICNC1 (input capture noise cancelling) can be set to 1 to filter inputs from the input capture pin. We haven't needed it.

Bit ICES1 (input capture edge select) selects the mode of the input capture bit: falling edge (0) or rising edge (1). We use this bit in the X8R PWM output detection, described later.

Bits WGM13 and WGM12 can be found in table 15-5 above. For mode 4 they are set to 0 and 1, respectively.

Bits CS12, CS11, CS10 can be found in the following table:

Table 15-6. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

If external pin modes are used for the Timer/Counter1, transitions on the T1 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

Now, some math. I have been describing the chopper as being at 10 Hz. That is true, but we actually want the timer/counter to compare match at 20 Hz, as it toggles the output of the clock generator chip (the signal source) on and off at every compare match. Essentially the signal needs to be on for 0.05 seconds and off for 0.05 seconds for a 10 Hz chopping frequency; this gives a timer/counter frequency of 20 Hz since it is working in those chunks of 0.05 seconds. The Arduino clock frequency is 16 MHz. Trying to set that to 20 Hz yields a top value for the counter of 16E6 / 20 = 800,000. This is well over the

maximum possible value of 65535. However, using a 1/64 clock prescaler yields a top value of (16E6 / 64) / 20 = 12500. This value is acceptable and 1/64 is the smallest prescaler that gives us a usable value (i.e. the best resolution). From benchtop measurements, a top value of 12508 most closely keeps a timer/counter overflow rate of 20 Hz. We set register OCR1A to this top value (see table 15-5). The top value of 12508 is off from the calculated value of 12500 (really 12499 due to how the timer/counter overflows). This is because the crystal oscillator of the Arduino is not precisely at 16 MHz; we measured its frequency to be 16.0114 MHz. The top count value should be tuned for the individual Arduino to set the appropriate timer/counter compare match frequency.

From the above table (15-6), for a 1/64 prescaler we set bits CS12, CS11, and CS10 to 0, 1, 1 respectively.

From all this, we set TCCR1B to an initial state of: B00001011.

3.2.2.3 Setting timer/counter interrupt mask register TIMSK1

From the datasheet[9], the bits of TIMSK1 that we need to set are:

15.11.8 TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

We need to enable input capture interrupts, and output compare match A interrupts. These are bits ICIE1 and OCIE1A, respectively. The other bits are described in the datasheet but aren't useful to us.

Thus, we set TIMSK1 to a state of: B00100010

3.2.2.4 Setting the data direction register DDRB

One final bitmask is set: the Data Direction Register. We want to set the data direction of the timer/counter outputs – OC1A and OC1B on pins 9 and 10. From the Arduino documentation[10], these pins fall under Port B of the ATMega328p chip:

> **PORTB** maps to Arduino digital pins 8 to 13 The two high bits (6 & 7) map to the crystal pins and are not usable
>
> DDRB - The Port B Data Direction Register - read/write
>
> PORTB - The Port B Data Register - read/write
>
> PINB - The Port B Input Pins Register - read only

From the ATMega328p datasheet[9], DDRB maps as follows:

13.4.3 DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Thus, to set the data direction of pins 9 and 10 to output, we set DDRB to B00000110 (i.e. decimal 6). It is functionally the same as setting pinMode of pins 9 and 10 to output.

3.3 Setting the states of the chopper

3.3.1 Overview

As has been discussed, we desire three states of the chopper, corresponding to the three positions of switch D on the Taranis:

Steady on (up)

Chopping, 10 Hz (middle)

Steady off (down)

We use the timer/counter compare match interrupt service routine to set the chopping behavior. We use the timer/counter input capture interrupt service routine to measure the PWM pulse width of the X8R output signal, which corresponds to Taranis switch position.

3.3.2 Timer/counter compare match interrupt service routine (used to set chopping cadence)

Setting the OCR1A register to 12508 with a 1/64 clock prescaler ensures that the timer/counter compare match interrupt happens at precisely 20 Hz. Within the compare match ISR, three global flags are used: on_flag, chop_flag, and off_flag. Only one of these Booleans is ever set to True at a time. When the interrupt happens and the ISR is triggered, one of three things will happen:

on_flag True: The output of clock 0 of the Si5351 eval board is enabled (only set when the ISR is first triggered after the on_flag is set to True, so that it is not being constantly re-enabled at 20 Hz).

chop_flag True: The output of clock 0 of the Si5351 eval board is toggled (i.e. if it was enabled, disables it and vice-versa). The output is off for one 20 Hz period and on for one 20 Hz period, giving a 10 Hz chopping period.

off_flag True: The output of clock 0 of the Si5351 eval board is disabled (only set when the ISR is first triggered after the off_flag is set to True, so that it is not being constantly re-disabled at 20 Hz).

The three flags are set in the main loop of the Arduino code according to the state of Taranis switch D, which is determined by reading the PWM output of the X8R radio.

3.3.3 Timer/counter input capture interrupt service routine (used to read PWM pulse width)

The input capture ISR is triggered whenever a rising or falling edge is detected at the ICP1 input pin (Arduino pin D8). Bit ICES1 of TCCR1B selects rising or falling edge detection. Whenever a capture event occurs, the ICR1 register is updated with the current timer count. When a rising event is detected, marking the beginning of a pulse, we store the timer count value as read from ICR1. Bit ICES1 is also flipped to detect the upcoming falling edge of the pulse. When the falling edge is detected, marking the end of the pulse, we again store the timer count value; bit ICES1 is also flipped again, set up to detect the upcoming rising edge of the next pulse. By subtracting the timer count at the beginning of the pulse from the timer count at the end of the pulse, we obtain the pulse width in units of timer counts. Overflows are accounted for by adding the maximum timer count value (12508) to the falling edge count value in the event the rising edge count value is greater than the falling edge count value. When a pulse is measured, a flag is set telling the main loop to store the measured pulse width.

The three positions of Taranis switch D correspond to three different PWM pulse widths output from the X8R receiver (approximate values):

1 ms (Taranis switch D up – corresponds to “steady on”)

1.5 ms (Taranis switch D middle – corresponds to “chopping”)

2 ms (Taranis switch D down – corresponds to “steady off”)

As the Arduino's clock is 16 MHz and we set the timer/counter with a 1/64 clock prescaler, these pulse widths correspond to timer counts of 250 (up), 375 (middle), and 500 (down). In practice we have measured count values of 247, 375, and 503. We think that these count values differ slightly from their calculated values due to measurements showing that the crystal oscillator of the Arduino does not have a frequency of precisely 16 MHz, and that the X8R PWM pulse widths are marginally off from the above time values. As all of these pulse widths are well below a 20 Hz period of .05 sec (i.e. maximum timer/counter value of 12508), we do not have to worry about multiple timer/counter overflows occurring in one pulse.

The main loop of the Arduino code detects the flag indicating a pulse has been measured. According to the measured pulse width (i.e. Taranis switch D position), it then sets the on_flag, chop_flag, and off_flag Booleans as appropriate, to be used in the compare match ISR.

4 Conclusion

Bench tests of this new payload seem to indicate a stable and usable output. We will fly it at Uapishka in March 2024 and report on any anomalous behavior.

5 References

1. Leo Bodnar: https://www.leobodnar.com/shop/index.php?main_page=product_info&products_id=301
2. HMC545 eval board: https://www.digikey.ca/en/products/detail/analog-devices-inc/EV1HMC545A/5403566
3. Arduino Mega: https://www.digikey.ca/en/products/detail/arduino/A000067/2639006
4. Aaronia BicoLog: https://aaronia.com/en/shop/radial-isotrope-antenne-bicolog-5070
5. Arduino (Abra) Nano: https://abra-electronics.com/robotics-embedded-electronics/arduino-boards/abranano-abra-arduino-nano-v3.0-compatible-atmega328p.html
6. Mikroe-4113 Si5351 eval board: https://www.digikey.ca/en/products/detail/mikroelektronika/MIKROE-4113/12318588
7. Adafruit Si5351 Arduino library: https://www.arduino.cc/reference/en/libraries/adafruit-si5351-library/
8. Etherkit Si5351 Arduino library: https://www.arduino.cc/reference/en/libraries/etherkit-si5351/
9. ATMega328p datasheet, specifically section 15 for timer/counter 1: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
10. Arduino port register documentation: https://docs.arduino.cc/retired/hacking/software/PortManipulation/