SparkFun logger

Hossam Marzouk

05, November 2020

Table of Contents

1 Logger

The SparkFun OpenLog Artemis is an open-source data logger that comes preprogrammed to automatically log IMU, GPS, serial data, and various pressure, humidity, and distance sensors.

1.1 Main features

- Operating voltage:
- 5V with USB type C
- 3.6V to 4.2V with LiPo battery
- Built-in MCP73831 single cell LiPo charger
- Current consumption ~33mA (running with GPS and ADC) and ~50mA with LCD lights on
- Built-in 4 channels ADC with 14-bit resolution and up to 1900Hz sampling rate
- SD card socket, Support FAT32 up to 32GB
- RTC with 1mAhr battery backup
- 9-axis IMU logging up to 250Hz

A detailed list of features and documentation can be found in the following links:

Table 1. SparkFun logger documentation links

Main page	https://www.sparkfun.com/products/16832
Schematic	https://cdn.sparkfun.com/assets/1/4/e/1/7/OpenLog_Artemis_9DoF_IMU_Schematic_v1_0_latest.pdf
Processor datasheet	https://cdn.sparkfun.com/assets/d/e/8/b/4/Apollo3_Blue_MCU_Data_Sheet_v0_12_1_rZ9Akgo.pdf
Hookup Guide	https://learn.sparkfun.com/tutorials/openlog-artemis-hookup-guide?_ga=2.77576021.1550399205.1604572944-63414889.1601900928

1.2 Firmware

The SparkFun logger comes preprogrammed with a firmware, current version 1.7, ready to auto-detect sensors, and start logging automatically to the SD card. The logging parameters can be configured by connecting to the logger with a USB cable and using a terminal window. Full description of firmware, up to date firmware version, and installation procedures can be found in the hookup guide link in Table 1.

1.3 Logger/firmware disadvantages

The original firmware gives a wide range of applications to use, but it is not very useful in high-frequency time precision applications such as telluric recording for the following reasons:

1- The logging starts immediately after power-on regardless of the GPS fix.
2- The GPS time is only used to update the RTC clock once and only upon booting.
3- The logging frequency is controlled by the RTC only.
4- No support to the LCD display.
5- The firmware states that the maximum achievable sample rate is 156Hz, but in reality, I couldn't get it to record more than 10 Hz due to a GPS update bug in the firmware.
6- Although they offer the option to log from an external trigger(i.e GPS), there is no way to change the default time pulses of the GPS from 1 PPS.
7- No gain control in the up to date version, but available in older versions!
8- The firmware uses built-in functions for sensors only, which limits the ADC to 20Hz if it is logging raw voltage. But it can go higher if it logs temperature!.
9- The durability of the SD card is very weak, it broke after a few times of removal to download the data and it is not repairable.
10- Editing the source code firmware is possible, but compiling takes a long time ~40 minutes.

1.4 Firmware Modification

I have started to fix most of the issues in Section 1.4 by doing the following edits to the firmware:

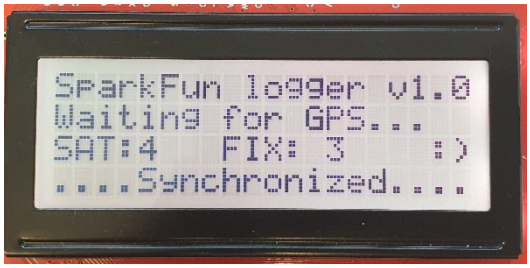
1- Decreased compiling time to ~2 minutes by removing all unused libraries and functions from the source code.

2- Forced the logger to wait for GPS fix before recording.

File	Lines
OpenLog_Artemis.ino	228:243; 381:394
menuAttachedDevices.ino	680:709

3- Display time, satellites, and fix type on the LCD.

File	Lines
OpenLog_Artemis.ino	365:379; 409:433
menuAttachedDevices.ino	680:709



4- Control the GPS time pule frequency and width (Missy code, but works)

File	Lines	Comment
autoDetect.ino	323:415	Need to manually change rate and ratio variables at line 324 (needs better function design)

I have stopped modification to that limit because I couldn't increase the max rate more than 10Hz and it's better to make new firmware instead of making ugly modifications to the original firmware.

2 GPS

2.1 Main features

- Internal antenna
- 72-Channel GNSS Receiver
- 18Hz Max Update Rate
- Time-To-First-Fix:
- Cold: 26s
- Hot: 1s
- Max G: ≤4
- Time Pulse Accuracy: 30ns

Table 2 SparkFun Ublox SAM M8Q GPS documentation links

Main page	https://www.sparkfun.com/products/15210
Schematic	https://cdn.sparkfun.com/assets/3/c/2/2/e/SparkFun_Ublox_SAM-M8Q.pdf
Datasheet	https://cdn.sparkfun.com/assets/4/e/b/9/f/SAM-M8Q_DataSheet__UBX-16012619_.pdf
u-blox Protocol Specification	https://cdn.sparkfun.com/assets/0/b/0/f/7/u-blox8-M8_ReceiverDescrProtSpec__UBX-13003221__Public.pdf
Hookup Guide	https://learn.sparkfun.com/tutorials/sparkfun-gps-breakout-zoe-m8q-and-sam-m8q-hookup-guide?_ga=2.114593863.1550399205.1604572944-63414889.1601900928

2.2 Advantages

- Powerful antenna, it can catch more than 7 satellites and fix of 3 indoors.
- Quick synchronization, max wait indoors 1 min while outdoors 20 seconds wait.
- Accurate time pulses.
- Back up internal oscillator for time pulse in case, the GPS lost sync.
- Different time grids to align time pules, UTC, GPS, GLONASS, BeiDou, and Galileo.

2.3 Disadvantages

- No build-in function to easily control the time pulse parameters, so I had to make my own functions that write/read binary commands to the GPS board (*…\Examples\PPS_control_Blink\PPS_ReadWrite.ino*).
- The documentation states that it can get a max G of 4, but in reality, I can only get what is called fix type of 3. Not sure if this is interpolated as G 4 or not. According to the example code provided by SparkFun, the following are the description of fix types:

Fix type	Description
0	no fix
1	Dead reckoning (requires external sensors)
2	2D (not quite enough satellites in view)
3	3D (the standard fix)
4	GNSS + dead reckoning (requires external sensors)
5	Time fix only

3 ADC

3.1 Main features

- Measurement Type: Two Differential (Default) or Four Single-Ended
- Programmable Gain Amplifier: x1 to x128
- Programmable Voltage Reference
- Internal Precision 2.048V Reference
- 24-bits (up to 20 bits effective resolution)
- Sample Rate: 20Hz to 2.0kHz
- At 20Hz the built-in digital filter provides simultaneous 50Hz and 60Hz noise rejection for industrial applications
- Current Consumption (Typical):
- 250µA to 955µA (depending on the selected mode)

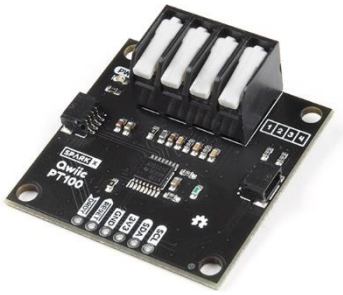


Table 3 SparkFun ADS122C04 ADC documentation links

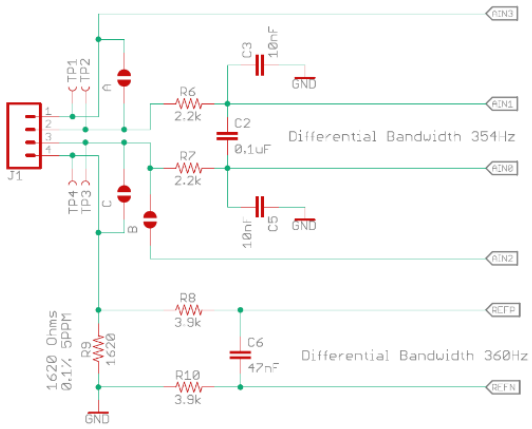
Main page	https://www.sparkfun.com/products/16770
Schematic	https://cdn.sparkfun.com/assets/9/b/6/d/8/Qwiic_PT100-Schematic.pdf
Datasheet	https://cdn.sparkfun.com/assets/7/4/e/1/4/ads122c04_datasheet.pdf

3.2 Advantages

- Different acquisition modes (2 wire,3 wire, 4 wire, and raw modes) to internally compensate for the cable resistance or to read the raw voltage value.
- Two operation modes(normal and turbo) to double the sample rate if needed.
- Two conversion modes (single and continuous) to control ADC conversion.
- Predefined sample rates functions of 20, 45, 90, 175, 330, 600, and 1000 Hz. If used in turbo mode the sample rate will be doubled.
- Predefined gain setting functions of 1, 2, 4, 8, 16, 32, 64, 128.

3.3 Disadvantages

- Unable to set the ADC to record between 2 differential channels at the same time, i.e the input channels can be 2 and 3 or 1 and 4 but not both as there are hardware jumpers must be switched to use each channel. It is hardware not possible.



- Unable to activate the internal sample counter.

- Not sure if I can change the sampling frequency to be as we want i.e 256 instead of 300.

- No ability to use external trigger directly to the ADC. i.e I have to trigger the logger which in terms triggers the ADC indirectly.

- Effective resolution is not great for high sample rate and gain, below is the effective resolution table from the datasheet(pages 18,19).

Table 2. Effective Resolution From RMS Noise (Noise-Free Resolution From Peak-to-Peak Noise) at AVDD = 3.3 V, AVSS = 0 V, Normal Mode, PGA Enabled, and Internal V_{REF} = 2.048 V

DATA RATE (SPS)	GAIN (PGA Enabled)							
	1	2	4	8	16	32	64	128
20	19.62 (17.53)	19.65 (17.54)	19.64 (17.44)	19.61 (17.22)	19.25 (17.36)	19.02 (16.99)	18.80 (16.85)	18.15 (16.09)
45	19.26 (17.06)	19.37 (17.11)	19.23 (17.20)	19.07 (16.94)	18.91 (16.68)	18.80 (16.49)	18.52 (16.46)	17.91 (15.78)
90	18.79 (16.59)	18.74 (16.36)	18.70 (16.51)	18.66 (16.23)	18.42 (16.18)	18.25 (15.92)	17.97 (15.70)	17.36 (15.10)
175	18.29 (15.97)	18.21 (15.74)	18.26 (15.88)	18.02 (15.48)	17.94 (15.57)	17.70 (15.23)	17.48 (15.02)	16.97 (14.39)
330	17.82 (15.21)	17.84 (15.35)	17.73 (15.12)	17.58 (15.15)	17.46 (14.96)	17.20 (14.75)	16.97 (14.41)	16.40 (14.03)
600	17.34 (14.70)	17.28 (14.72)	17.20 (14.68)	17.02 (14.70)	16.95 (14.37)	16.73 (14.22)	16.46 (13.80)	15.94 (13.32)
1000	16.76 (14.13)	16.79 (14.20)	16.72 (14.10)	16.51 (13.99)	16.45 (13.99)	16.24 (13.61)	15.91 (13.33)	15.42 (12.74)

Table 6. Effective Resolution From RMS Noise (Noise-Free Resolution From Peak-to-Peak Noise) at AVDD = 3.3 V, AVSS = 0 V, Turbo Mode, PGA Enabled, and Internal V_{REF} = 2.048 V

DATA RATE (SPS)	GAIN (PGA Enabled)							
	1	2	4	8	16	32	64	128
40	19.83 (17.69)	19.80 (17.56)	19.80 (17.55)	19.63 (17.51)	19.44 (17.25)	19.15 (16.83)	18.91 (16.63)	18.29 (15.94)
90	19.44 (17.02)	19.39 (17.14)	19.27 (16.99)	19.09 (16.82)	18.91 (16.59)	18.66 (16.30)	18.44 (16.01)	17.70 (15.36)
180	18.88 (16.49)	18.80 (16.49)	18.77 (16.24)	18.54 (16.19)	18.46 (15.93)	18.18 (15.65)	17.80 (15.37)	17.15 (14.90)
350	18.29 (15.82)	18.30 (15.76)	18.23 (15.71)	18.05 (15.55)	17.91 (15.40)	17.68 (15.14)	17.25 (14.87)	16.75 (14.25)
660	17.87 (15.24)	17.79 (15.19)	17.79 (15.17)	17.46 (14.88)	17.46 (14.89)	17.18 (14.62)	16.78 (14.23)	16.25 (13.71)
1200	17.31 (14.70)	17.32 (14.67)	17.25 (14.66)	17.00 (14.39)	16.89 (14.24)	16.67 (14.07)	16.27 (13.60)	15.75 (13.16)
2000	16.82 (14.14)	16.81 (14.15)	16.73 (14.07)	16.54 (13.92)	16.37 (13.74)	16.15 (13.49)	15.80 (13.15)	15.23 (12.53)

4 Firmware Modification Guide

The SparkFun logger is a very powerful microcontroller but the main disadvantage that it comes with predefined firmware and according to the SparkFun staff it is NOT meant to be modified or edited by any other tool.

Despite the logger not meant to be edited, it is still an open-source firmware and some workarounds can be made to edit or make new firmware.

In the following sections, I will make a step by step guide on how to install the right tools to start editing and how to program each component to work separately and combined with other components. I will also provide some examples programmed in Arduino IDE.

4.1 Installation guide

The first and most important tool needed in programming microcontrollers is the right compiler, in this case, we will use the Arduino IDE compiler.

Most of the following instructions are from this page https://learn.sparkfun.com/tutorials/artemis-development-with-the-arduino-ide/all, but unfortunately, not everything is compatible with this specific logger so, I will make my own detailed instructions.

Install Arduino IDE

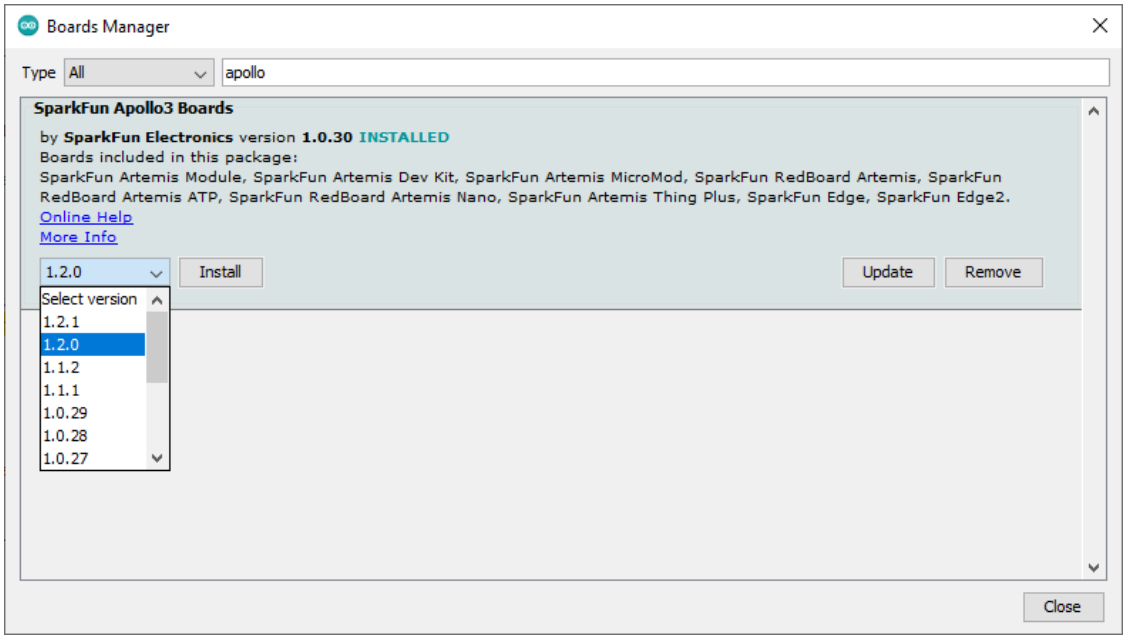
According to the SparkFun documentation, it is best to use the *Arduino IDE version 1.8.12,* but I found no problem at all using the up to date version 1.8.13 which can be downloaded from here: https://learn.sparkfun.com/tutorials/artemis-development-with-the-arduino-ide/all

4.1.1. Install SparkFun Apollo3 Boards

Users will need to install the **SparkFun Apollo3 Boards** board definitions in the Arduino IDE for our development boards with an Artemis module

1- From File>Preferences, copy the following URL to the Additional boards manager https://raw.githubusercontent.com/sparkfun/Arduino_Boards/master/IDE_Board_Manager/package_sparkfun_index.json

2- Tools>Boards>Boards Manager; Type in search Apollo3 then install SparkFun Apollo3 boards. Make sure the **version is 1.2.0** or older, **NEVER install new versions**



3- Tools>Boards>SparFun Apollo3; choose **SparkFun RedBoard Artemis ATP**, **NEVER choose another board.**

4- Tools>Boards>BootLoader; choose **SparkFun Variable Loader**, **NEVER choose another loader.**

5- Finally, don't forget to set the Port and baud rate according to your configuration.

4.1.2. Install Libraries

All the required libraries for the SparkFun logger to work can be found in this link: https://github.com/hossammarzouk/SparkFun-Logger/blob/main/libraries.zip

Just download all the libraries and copy to the directory **"....\Documents\Arduino\libraries".**

I have included a lot of unused libraries in the link, in case someone compiles the original firmware without deleting unused functions.

4.2 Examples

In the following section, I have provided some examples to use with the SparkFun from a simple blink example to read ADC and GPS data.

I will not be able to write all examples here as some require multiple pages, But I will add some remarks here on how some codes work and usefull tricks.

All the example codes and maybe can be found in the following GitHub link:
https://github.com/hossammarzouk/SparkFun-Logger

4.2.1. Blinking

The purpose of this example to make sure we had established a connection between logger and PC by turning the power and status LEDs on and off.

Note: Most of the predefined blink example will not work on the SparFun logger because it doesn't have LED_BUILTIN defined so, the predefinition of the LEDs pin fix this issue.

```
*/ This is blink example
const byte power_led  = 29;
const byte status_led = 19;

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(power_led, OUTPUT);
  pinMode(status_led, OUTPUT);

}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(power_led, HIGH);     // turn the LED level
  digitalWrite(status_led, LOW);     // turn the LED LOW
  delay(500);                       // wait for a second

  digitalWrite(status_led, HIGH);    // turn the LED LOW
  digitalWrite(power_led, LOW);      // turn the LED onlevel)
  delay(500);
  }
```

4.2.2. LCD HelloWorld

In this example, we test the I2C connection between one or multiple devices, which we will use this sequence with any device attached to the logger.

- Most of the I2C communication uses the library <*Wire.h*>, but it is not compatible with the logger because the logger uses Wire1.h. The workaround is to define `TwoWire qwiic(1)` at the start and call it from now on instead of *Wire.*
- All the I2C devices are connected together through qwiic connector started in 3 stages :
    - Power up everything, (`pinMode(18, OUTPUT); digitalWrite(18, HIGH);`)
    - Start all qwiic channels (`qwiic.begin();`)
    - Start communicating with specific device (`lcd.begin(qwiic);`)

```
#include <Wire.h> //Needed for I2C to lcd
TwoWire qwiic(1); //Needed to use Wire1 instead of Wire

#include <SerLCD.h> http://librarymanager/All#SparkFun_SerLCD
SerLCD lcd;

int i = 0; // loop counter

void setup()

{
  Serial.begin(115200);

  pinMode(18, OUTPUT);
  digitalWrite(18, HIGH); // power up the qwiic connector

  delay(100);          // need several delays for lcd to start, not sure why!!!
  qwiic.begin();       // start all qwiic channels
  lcd.begin(qwiic);    // Make sure gps is started with correct address
  delay(1000);         // need several delays for lcd to start, not sure why!!!

  Serial.print("LCD example");

  lcd.print("Hello World :)");


}
```

4.2.3. GPS get time and position

We use the same previous method in initializing the sensor:

- Define connection library, `TwoWire qwiic(1);`
- Power up everything, (`pinMode(18, OUTPUT);  digitalWrite(18, HIGH);`)
- Start all qwiic channels (`qwiic.begin();`)
- Start communicating with specific device (`myGPS.begin(qwiic);`)

```
#include <Wire.h> //Needed for I2C to GPS
TwoWire qwiic(1); //Needed to use Wire1 instead of Wire

#include "SparkFun_Ublox_Arduino_Library.h"
/http://librarymanager/All#SparkFun_Ublox_GPS

SFE_UBLOX_GPS myGPS;

void setup()
{
  Serial.begin(115200);

  Serial.println("SparkFun Ublox Example");

  pinMode(18, OUTPUT);
  digitalWrite(18, HIGH); // power up the qwiic connector

  qwiic.begin();       // start all qwiic channels
  myGPS.begin(qwiic);  // Make sure gps is started with correct address

  myGPS.setI2COutput(COM_TYPE_UBX); //Set the I2C port to output UBX only (turn
off NMEA noise)

}

}
```

4.2.4. ADC read raw voltage

Again, the same starting sequence but this time we need to specify the ADC I2c address. The default address is 0x45, then we can begin the sensor by `(mySensor.begin(address_ADC, qwiic)).`

We must also configure the working mode of ADC between raw,2wire,3wrire and 4wire. In this example we use the raw mode to read raw voltage values.

Finally, a conversion must be made to convert the raw values to volts by dividing the value to the LSB. The LSB is 2.048 / 2^23 = 0.24414 uV (0.24414 microvolts).

```
#include <SparkFun_ADS122C04_ADC_Arduino_Library.h> // Click here to get the
library: http://librarymanager/All#SparkFun_ADS122C0
#include <Wire.h>
TwoWire qwiic(1); //Needed to use Wire1 instead of Wire
SFE_ADS122C04 mySensor;
#define  address_ADC   0x45   // default address of ADC
int i = 0;
void setup(void)
{

  pinMode(18, OUTPUT);
  digitalWrite(18, HIGH); // power up the qwiic connector

  qwiic.begin();       // start all qwiic channelsdelay(100);

  Serial.begin(115200);
  while (!Serial)
    ; //Wait for user to open terminal
  Serial.println(F("Qwiic PT100 Example"));

  //mySensor.enableDebugging(); //Uncomment this line to enable debug messages
on Serial
  mySensor.begin(address_ADC, qwiic);

  mySensor.configureADCmode(ADS122C04_RAW_MODE); // Configure the PT100 for raw
mode
  mySensor.setDataRate(ADS122C04_DATA_RATE_600SPS); // try to increase sampling
freq, but failed!!
}

void loop()
{
  // Get the raw voltage as int32_t
  int32_t raw_v = mySensor.readRawVoltage();

  // Convert to Volts (method 1)
  float volts_1 = ((float)raw_v) * 244.14e-9;

  // Convert to Volts (method 2)
  float volts_2 = ((float)raw_v) / 4096000;
}
```

4.2.5. PPS Configuration

There is no preset functions to control the PPS in the GPS modeule so, we have to send commands to the GPS in binary to change the default configuration.

According to the U-blox8 protocol page 67 (link in section 3.1), the following parameters can be used to control PPS:

- **time pulse index** - Index of time pulse output pin to be configured
- **antenna cable delay** - Signal delay due to the cable between antenna and receiver.
- **RF group delay** - Signal delay in the RF module of the receiver (read-only).
- **pulse frequency/period** - Frequency or period time of the pulse when locked mode is not configured or active.
- **pulse frequency/period lock** - Frequency or period time of the pulse, as soon as receiver has calculated a valid time from a received signal. Only used if the corresponding flag is set to use another setting in locked mode.
- **pulse length/ratio** - Length or duty cycle of the generated pulse, either specifies a time or ratio for the pulse to be on/off.
- **pulse length/ratio lock** - Length or duty cycle of the generated pulse, as soon as receiver has calculated a valid time from a received signal. Only used if the corresponding flag is set to use another setting in locked mode.
- **user delay** - The cable delay from the receiver to the user device plus signal delay of any user application.
- **active** - time pulse will be active if this bit is set.
- **lock to gps freq** - Use frequency gained from GPS signal information rather than local oscillator's frequency if flag is set.
- **lock to gnss freq** - Use frequency gained from GNSS signal information rather than local oscillator's frequency if flag is set.
- **locked other setting** - If this bit is set, as soon as the receiver can calculate a valid time, the alternative setting is used. This mode can be used for example to disable time pulse if time is not locked, or indicate lock with different duty cycles.

- **is frequency** - Interpret the 'Frequency/Period' field as frequency rather than period if flag is set.
- **is length** - Interpret the 'Length/Ratio' field as length rather than ratio if flag is set.
- **align to TOW** - If this bit is set, pulses are aligned to the top of a second.
- **polarity** - If set, the first edge of the pulse is a rising edge (Pulse Mode: Rising).
- **grid UTC/GPS** - Selection between UTC (0) or GPS (1) timegrid.
- **grid UTC/GNSS** - Selection between UTC (0), GPS (1), GLONASS (2) and Beidou (3) timegrid.

According to the U-blox8 protocol page 222 (link in section 3.1), the following are the bytes index and format of each parameter:

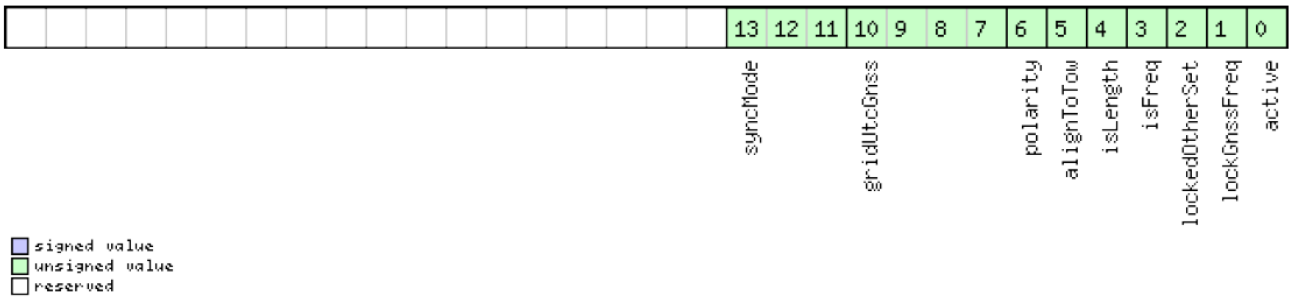
Byte Offset	*Number Format*	*Scaling*	*Name*	*Unit*	*Description*
0	U1	-	`tpIdx`	-	Time pulse selection (0 = TIMEPULSE, 1 = TIMEPULSE2)
1	U1	-	`version`	-	Message version (0x00 for this version)
2	U1[2]	-	`reserved1`	-	Reserved
4	I2	-	`antCableDelay`	ns	Antenna cable delay
6	I2	-	`rfGroupDelay`	ns	RF group delay
8	U4	-	`freqPeriod`	Hz_or_us	Frequency or period time, depending on setting of bit 'isFreq'
12	U4	-	`freqPeriodLock`	Hz_or_us	Frequency or period time when locked to GPS time, only used if 'lockedOtherSet' is set
16	U4	-	`pulseLenRatio`	us_or_2^-32	Pulse length or duty cycle, depending on 'isLength'
20	U4	-	`pulseLenRatioLock`	us_or_2^-32	Pulse length or duty cycle when locked to GPS time, only used if 'lockedOtherSet' is set
24	I4	-	`userConfigDelay`	ns	User configurable time pulse delay

CFG-TP5 continued

Byte Offset	*Number Format*	*Scaling*	*Name*	*Unit*	*Description*
28	X4	-	`flags`	-	Configuration flags (see graphic below)

Bitfield flags

This graphic explains the bits of `flags`



Name	*Description*
`active`	if set enable time pulse; if pin assigned to another function, other function takes precedence
`lockGpsFreq`	if set synchronize time pulse to GPS as soon as GPS time is valid, otherwise use local clock
`lockedOtherSet`	if set use 'freqPeriodLock' and 'pulseLenRatioLock' as soon as GPS time is valid and 'freqPeriod' and 'pulseLenRatio' if GPS time is invalid, if flag is cleared 'freqPeriod' and 'pulseLenRatio' used regardless of GPS time
`isFreq`	if set 'freqPeriodLock' and 'freqPeriod' interpreted as frequency, otherwise interpreted as period
`isLength`	if set 'pulseLenRatioLock' and 'pulseLenRatio' interpreted as pulse length, otherwise interpreted as duty cycle
`alignToTow`	align pulse to top of second (period time must be integer fraction of 1s)
`polarity`	pulse polarity: 0 = falling edge at top of second 1 = rising edge at top of second
`gridUtcGps`	timegrid to use: 0 = UTC 1 = GPS

For the example code below to read the default configuration of the PPS we need to do the following:

1- Declare empty array to hold configuration

2- Read the binary configuration

3- Decode each parameter in its index bytes according to its format

The multi bytes parametres need to be decoded by, shifting all the adjancent bits to be aligned with the first 8 bits then sum up. For example, the freqperiod parameters which is 4 bytes long from(8:11), we read the last byte shift it by 3 bytes (8*3 bits), then read the second to last bytes shift it by 8*2 bits, then read second bytes and shift 1 byte(8 bits) and finally read the first byte with no shifting and sum all bits together.

```
bool ReadPPS() {
  uint8_t customPayload[MAX_PAYLOAD_SIZE]; // This array holds the payload data

  // The next line creates and initialises the packet information which wraps
around the payload
  ubxPacket customCfg = {0, 0, 0, 0, 0, customPayload, 0, 0,
SFE_UBLOX_PACKET_VALIDITY_NOT_DEFINED, SFE_UBLOX_PACKET_VALIDITY_NOT_DEFINED};

  customCfg.cls = UBX_CLASS_CFG; // This is the message Class
  customCfg.id = UBX_CFG_TP5; // This is the message ID
  customCfg.len = 0; // Setting the len (length) to zero let's us poll the
current settings
  customCfg.startingSpot = 0; // Always set the startingSpot to zero (unless
you really know what you are doing)

  // We also need to tell sendCommand how long it should wait for a reply
  uint16_t maxWait = 1100; // Wait for up to 250ms (Serial may need a lot
longer e.g. 1100)

  if (myGPS.sendCommand(&customCfg, maxWait) != SFE_UBLOX_STATUS_DATA_RECEIVED)
{
    Serial.println(F("Error reading pps ")); return (false);}

  // global varaiables (datatypes obtainded from ublox proctocol page 222)
uint8_t  tpIdx, Version;
int16_t  antCableDelay, rfGroupDelay;
uint32_t freqPeriod = 5, freqPeriodLock, pulseLenRatio, pulseLenRatioLock;
int32_t  userConfigDelay;
byte     flags;

  tpIdx               =  customPayload[0];
  Version             =  customPayload[1];
  antCableDelay       = (customPayload[5]  << 8 ) | (customPayload[4]);
  rfGroupDelay        = (customPayload[7]  << 8)  | (customPayload[6]);
  freqPeriod          = (customPayload[11] << 8 * 3) | (customPayload[10] << 8
* 2) | (customPayload[9]  << 8) | customPayload[8];

  freqPeriodLock      = (customPayload[15] << 8 * 3) | (customPayload[14] << 8
* 2) | (customPayload[13] << 8) | customPayload[12];

  pulseLenRatio       = (customPayload[19] << 8 * 3) | (customPayload[18] << 8
* 2) | (customPayload[17] << 8) | customPayload[16];

  pulseLenRatioLock   = (customPayload[23] << 8 * 3) | (customPayload[22] << 8
* 2) | (customPayload[21] << 8) | customPayload[20];

  userConfigDelay     = (customPayload[27] << 8 * 3) | (customPayload[26] << 8
* 2) | (customPayload[25] << 8) | customPayload[24];

  flags               =  customPayload[28];

}
```

The following example code shows how to change the default PPS paramertes **after** reading the configuration:

1- Declare empty array to hold configuration

2- Read the default configuration

3- Change desired parameter in the same way that we used in reading

4- Write all configuration back to GPS

```
byte Flags = customPayload[28];
bitWrite(Flags, 3, 1); // change frequency unit from period to frequency
//bitWrite(Flags, 4, 0); // change duration unit from lenghth to ratio
customPayload[28] = Flags;
//.............. Done setting flags.............

//.............. start setting frequency.............

//update freqPeriodLock
customPayload[12] = rate & 0xFF; // Store it in the payload
customPayload[13] = rate >>  8;
customPayload[14] = rate >> 8 * 2;
customPayload[15] = rate >>  8 * 3;

//update freqPeriod (we use only freq lock but update this just in case)
customPayload[8]  = rate & 0xFF; // Store it in the payload
customPayload[9]  = rate >>  8;
customPayload[10] = rate >> 8 * 2;
customPayload[11] = rate >>  8 * 3;

//.............. Done setting frequency.............

if (myGPS.sendCommand(&customCfg, maxWait) != SFE_UBLOX_STATUS_DATA_SENT)
{
 return(false);
 Serial.print("Error updating pps");while(1);
      }
```