SYNCHRONIZATION OF ONE PPS SIGNAL ON LINUX ENVIRONMENT

Joshi Ronak Narendrakumar
M.Tech In Embedded & VLSI Systems
U.V.Patel college of engineering, Ganpat University, Gujarat, India

Abstract--- As in present scenario, GPS is a very popular device among users for tracking and navigation purpose. This tracking and navigation procedure is based on many GPS receivers. Mostly all GPS receivers provides one PPS (Pulse per second) signal which is most required for synchronizing procedures. In many GPS receivers 1 PPS signal can be generated using spacial dedicated commands which is given to receiver and according to that command, 1 pps signal can be generated at the dedicated pin of the receiver. This paper is focused on the Generation of one PPS signal in the Linux environment. In the Linux operating system the same one pps signal is generated using specific daemon program based on Network Time Protocol and gpsd.

Keywords:- 1 PPS, daemon program, ntpd, gpsd

I. INTRODUCTION

Many GPS receivers provide a timing pulse, the so-called 'one pulse per second'(1PPS) signal. This pulse normally has a rising edge aligned with the GPS second, and can be used to discipline local clocks to maintain synchronization with Universal Time (UT). The characterization of the 1PPS timing pulse from GPS receivers can provide useful information to anyone contemplating using GPS for timing applications. 1 PPS signal is the TTL level pulse with a width of 200ms isolated output coming from the GPS receiver. The rising edge of the 1PPS signal is considered as the time reference. The falling edge will occur approximately 200 ms (_ 1 ms) after the rising edge. The falling edge will not be used for accurate time keeping. A simple timing diagram of 1PPS output signal timing is shown in figure 1.

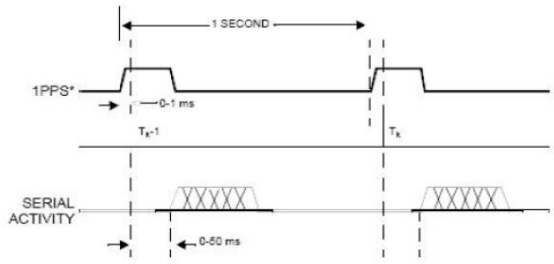


Fig. 1: PPS Output Signal Timing[1]

The Position, Status, Data message and time RAIM setup are the only output messages containing the time information. After the rising edge of 1 PPS signal, these message will be output from the receiver shortly. Generally the first data byte in the first message will be output between 0 to 50 ms after the rising edge of 1 PPS signal as shown in Figure 1. The position, status and data message will be reflected at the output port after the first data byte.

The Pulse Per Second (PPS) signal generated by a GPS receiver is used for the synchronization between separate stations in the communications system or the measurement system. The PPS signal can be observed in excellent relative accuracy because of followings. Firstly, the time difference among GPS satellites which influences the accuracy of PPS signal is controlled on the ground. Secondly, in the short-range synchronization commonly used in the communication-service cell, the same GPS satellites can be observed simultaneously. The above-mentioned utilization of GPS time information enables the accurate relative synchronization; currently the PPS accuracy in the GPS receiver is about 20 nanoseconds, which is not the limit of the accuracy of PPS signal.

II. GPSD DEAMON IN UBANTU

The Data – Logger signal conditioner card is a 16-bit, universal input module that features programmable input ranges on all channels. This module is an extremely cost-effective solution for industrial measurement and monitoring applications. Its opto-isolated inputs provide 1,000 VDC of isolation between the analog input and the module, protecting the module and peripherals from damage due to high input line voltage.

GPSD is a combination of tools for managing collections of GPS devices and other sensors related to navigation and precision timekeeping. The main program (daemon) manages a collection of sensors and makes report on a well known TCP/IP port.

In embedded system environment gpsd is used for navigation, precision agriculture, location-sensitive scientific telemetry and network time service. GPSD supports precision time keeping as it can act as a time source for ntpd if any of its attached sensors have PPS capability. The software layer of GPSD is divided into four parts: the drivers, the packet sniffer, the core library and the multiplexer. The structure of software layer is as follows:

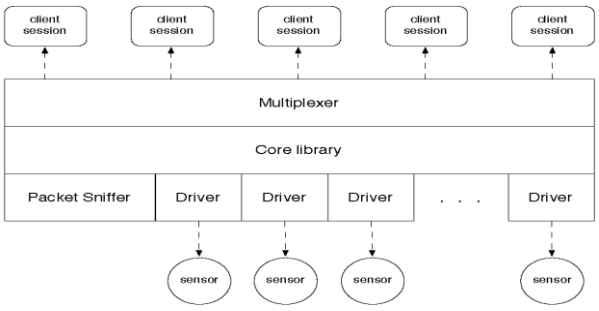


Fig. 2: Software Layer of GPSD

The driver is working as user space device driver for each sensors. The primary methods supports to parse a data packet into time, position ,velocity and status information, change its mode or baud rate, probe for device subtype etc. An auxiliary method may support driver control operations, such as changing the serial speed of the device. The packet sniffer is used for mining data packets out of

serial data stream as checksum algorithm. The core library works as follows:

starting a session by opening the device and reading data from it, verify the baud rate and parity/stop bit combination

until packet sniffer achieves synchronization lock with a known packet type.

- polling the device for a packet.
- closing the device and wrapping up the session.

The core library is responsible for switching each GPS connection to the correct device driver depending on the packet type that the sniffer returns.

The core library may change by time, if the device switches between different protocols. The multiplexer is used to handle client sessions and device management. It is responsible for passing reports to the clients, accepting client commands.

In normal operation, GPSD accepts input from one of the following source and processes in a loop as follows:

- A set of clients making requests through a TCP/IP port.
- A set of navigation sensors connected through serial or USB devices.
- A special control socket used by hotplug scripts or some configuration tools

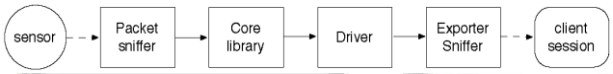


Fig. 3: Function Data flow of GPSD

When any of the input device is detected at the input port, the multiplexer device will put the device in its internal list of sensors. when client sends a watch request, the multiplexer layer opens the navigation sensors in its list and begins accepting data from them. Otherwise all GPS devices are closed and the daemon will be closed. Devices that stop sending data get timed out from the list. When data received in form of packet from any of the above sources, it is fed to the packet sniffer. The job of packet sniffer is to accumulate data from each port and recognizing, when it has accumulated a packet of a known type. After accumulate the packet, the core library carry the packet to the driver associated with its type. The driver's job is to separate data from the packet and payload into a per device session structure and set some status bit to indicate the multiplexer layer that what kind of data it got. From the status bit, one special dedicated bit indicates that the daemon has accumulated enough data to report to the client. When this bit is raised, it means it indicates end of packets. So the data in the device's session structure should be passed to one of the exporters. The exporter generates a report object and send it to all clients watching the device. Here a shared-memory exporter is available that copies the data to a shared memory segments.

A. Interfacing Of Gpsd Deamon With GPS Receiver

If you have a GPS attached on the lowest numbered USB port of a Linux system, and want to read reports from it on TCP/IP port 2947, it will normally suffice to do this:

gpsd /dev/ttyUSB0

For the lowest-numbered serial port:

gpsd /dev/ttyS0

Change the device number as appropriate if you need to use a different port. Command-line flags enable verbose logging, a control port, and other optional extras but should not be needed for basic operation; the one exception, on very badly designed hardware, might be -b (which see). On Linux systems supporting udev, gpsd is normally started automatically when a USB plugin event fires (if it is not already running) and is handed the name of the newly active device. In that case no invocation is required at all. For your initial tests set your GPS hardware to speak NMEA, as gpsd is guaranteed to be able to process that. If your GPS has a native or binary mode with better performance that gpsd knows how to speak, gpsd will auto configure that mode.

B. Procedure For Taking Time By Ntp Daemon From Gpsd

gpsd can provide reference clock information to ntpd, to keep the system clock synchronized to the time provided by the GPS receiver. If you're going to use gpsd you probably want to run it -n mode so the clock will be updated even when no clients are active.

Note that deriving time from messages received from the GPS is not as accurate as you might expect. Messages are often delayed in the receiver and on the link by several hundred milliseconds, and this delay is not constant. On Linux, gpsd includes support for interpreting the PPS pulses emitted at the start of every clock second on the carrier-detect lines of some serial GPSes; this pulse can be used to update NTP at much higher accuracy than message time provides. You can determine whether your GPS emits this pulse by running at -D 5 and watching for carrier-detect state change messages in the logfile. In addition, if your kernel provides the RFC 2783 kernel PPS API then gpsd will use that for extra accuracy.

When gpsd receives a sentence with a timestamp, it packages the received timestamp with current local time and sends it to a shared-memory segment with an ID known to ntpd, the network time synchronization daemon. If ntpd has been properly configured to receive this message, it will be used to correct the system clock.

III. NETWORK TIME PROTOCOL IN UBUNTU

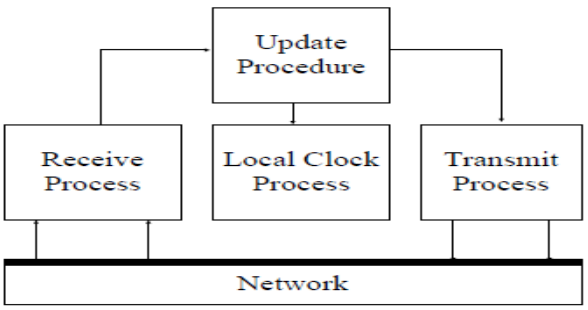


Fig. 4: Implementation Model Of NTP

In NTP time can be synchronized by passing the time information from one time source to another, starting from reference clock which is connected to stratum 1 server. As stratum 1 server is synchronized, it will synchronized stratum 2 server. For synchronizing the client with the server consists of several packet exchanges where each exchange in form of request and reply. When client sending the request, it will store its own time (Originate Time stamps) into the packet being sent. When server receive this packet, it will store its own time 4 (Receive Time stamps)

into the packet and packet will be returned after putting Transmit Time Stamp into the packet. The time difference can be used to estimate the time offset between client and server.

The shorter the time difference is more accurate to the server.

As shown in implementation model, the transmit process collects information in the data base and sends NTP message to the clients. Each message contain the local time stamps, receive time stamp and other information. The receive time stamps receives NTP message as well as information regarding radio clocks. Here the offset is calculated using peer clock and local clock. The update procedure collects offset data from each peer and select the best one using the algorithm. The local clock procedure adjust the phase and frequency according to offset value.

The ntpd utility is an operating system daemon which sets and maintains the system time of day in synchronism with Internet standard time servers. It has a complete implementation of Network Time protocol version 4. The ntpd utility does it most computation in 64-bit floating point arithmetic and floating point arithmetic with the precision of about 232 picoseconds. ntpd reads the configuration file at start up time in order to determine the synchronization source and operating modes.

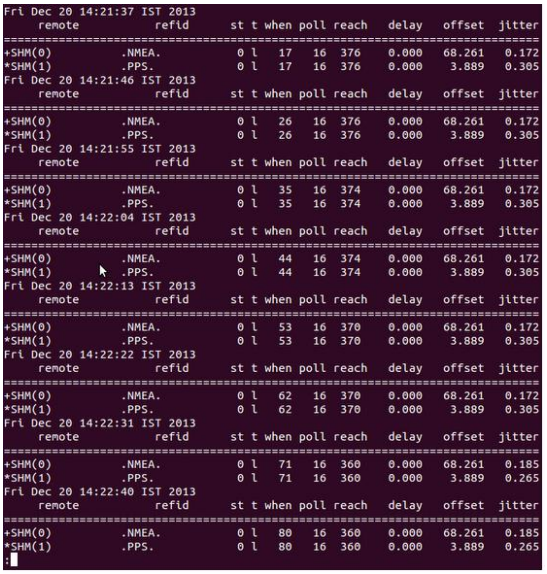
IV. Experimental Setup

- This implementation is tested in UBANTU 12.10 operating system. Steps for the procedure are as follows:
- Install NTPD deamon on the system by
  sudo apt-get install ntp
- Install GPSD deamon on the system by
  sudo apt-get install gpsd
- change ntp.conf file as follows:
  server 127.127.28.0 minpoll 4 maxpoll 4
  fudge 127.127.28.0 time1 0.420 refid
- NMEA
  server 127.127.28.1 minpoll 4 maxpoll 4
- Prefer fudge 127.127.28.1 refid PPS
- Procedure to restart NTPD & GPSD by following
  Commands restart NTPD:
- /etc/init.d/ntp restart
  - restart GPSD:
- /etc/init.d/gpsd restar

V. Conclusion

Using NTP with GPSD we can get more accurate time signal for our system. By adding 1 PPS source in the system make the very high accurate system. For getting the high accurate time up to 1 nano second this pps signals are used

VI. Test Results



References

[1] http://www.gmat.unsw.edu.au/snap/publications/mumford 2003a.pdf

