Cosimo Della Santina

"GPS Bc637PCIe:study about synchronization possibilities and implementations"

Topics of work

- Bc637PCIe (GPS)

One of the most used GPS functionality is the possibility of synchronization.

- ✓ Synchronization of the system clock with UTC

- ✓ Use interrupts to execute a code periodically

Topics of work 2

My work was to study these topics, account pro and cons, write some sample code that can be used in future to implements synchronized systems

Synchronization in MicroBoone

In MicroBoone experiment the data are acquired in frames. Frames are 1.6 msec long with data sampled at 2MHz. But the DAQ clock runs at a frequency of 16 MHz.

We need to know the GPS time corresponding to the frame number NOW.

The effect is that we have in memory a 3-column table of GPS time, DAQ Clock Time, Frame Number, as many rows as seconds that we want to keep in a circular buffer.

Clock synchronization

The main way to obtain clock synch. Is using NTP protocol.

Network Time Protocol (NTP) is a networking protocol for clock synchronization between computer systems over packet-switched, variable-latency data networks (such as internet)

NTP and PTP

NTP is particularly convenient with FermiLinux because, being a RedHat fork, has already implemented NTP demon.

On the other hand PTP, an evolution of NTP protocol, ensures better performance, but ad hoc switch are needed.

Interrupt + GPS

As we have already said, GPS generate a periodic signal that is called Pulse Per Second

It is possible to see this signal as an event, and to associate to it an appropriate routine through the mechanism of interruption.

Interrupt + GPS 2

In this way we can obtain a periodic routine, with a period accurate at nanoseconds

It is possible to implements periodic routine also using timer (I've done it in the consumer, see later)

Difference PPS/Timer

There are two important advantages in use PPS+interrupts implementation

- PPS is strongly more accurate than Timer

- PPS is Synchronized with Coordinated Universal Time. Therefore, for example, two different hosts, using PPS, execute their routine at the same time.

My codes

I've wrote some sample codes that implement this mechanism, called fancifully:

- GPS_1
- GPS_1_5
- GPS_2
- connection_control

GPS_1

This code include a main that sets up the card and the OS Interrupt table, and an interrupt routine, associated with PPS event, that reads directly time and print it.

The role of this is just demonstrate how to write and set a periodic PPS code.

GPS_1 code: main

Main calls initialization functions and wait:

- bcStartPci(); sets and starts the device
- pci_set_ints(hBC_PCI); sets Interrupt Descriptor Table placing handler routine in it (bcStartIntEx(hBC_PCI, bcIntHandlerRoutine, INTERRUPT_1PPS & 0x7F))

GPS_1 code: interrupt

Interrupt code:

```
//Get current time

// bcReadDecTimeEx is a library function that reads current time

bcReadDecTimeEx (hBC_PCI, &dectime, &min, &nano, &stat);


//Print time read

printf("pps_routine: %02d:%02d:%02d.%06lu%d \n",

    dectime.tm_hour, dectime.tm_min, dectime.tm_sec, min,
  nano);
```

GPS_1 output

OUTPUT:
...
bcIntHandlerRoutine
pps_routine: 80:156:66.0040402
bcIntHandlerRoutine
pps_routine: 80:156:67.0140657
bcIntHandlerRoutine
pps_routine: 80:156:68.0030917
bcIntHandlerRoutine
pps_routine: 80:156:69.0041193
bcIntHandlerRoutine
pps_routine: 80:156:70.0041424
bcIntHandlerRoutine
pps_routine: 80:156:71.0041674
bcIntHandlerRoutine
pps_routine: 80:156:72.0141917
bcIntHandlerRoutine
pps_routine: 80:156:73.0052188
bcIntHandlerRoutine
pps_routine: 80:156:74.0032431
bcIntHandlerRoutine
pps_routine: 80:156:75.0032680
bcIntHandlerRoutine
pps_routine: 80:156:76.0022934
...

This output demonstrate that:

- Routine occurs each second

- Time is captured with a random delay, due by the execution of software between interrupt signal and data acquisition.

GPS_1_5

This code is similar to the previous (GPS_1). The difference lies in how the current time is obtained: in this new implementation we use event register, latching (on hardware level) the time in which PPS arrives in it.

This code demonstrate how to latch an event (that could be different from PPS) time

Code differences between 1 and 1_5

Into main I need to setup event register:

```
iVal=1;

EvDat.evtsrc = (BYTE)iVal;

EvDat.evtctl = (BYTE)iVal;

iVal=0;

EvDat.evtlock = (BYTE)iVal;

EvDat.evtsense = (BYTE)iVal;

bcSetEventsData (hBC_PCI, &EvDat);
```

Into interrupt routine I only need to take time with another function:

```
bcReadEventTimeEx (hBC_PCI, &evtmaj, &evtmin, &evtnano, &stat);
```

GPS_1_5 output

OUTPUT:

...
Time: 09/13/2012 17:01:07.0000000   Status: 7
Time: 09/13/2012 17:01:08.0000000   Status: 7
Time: 09/13/2012 17:01:09.0000000   Status: 7
Time: 09/13/2012 17:01:10.0000000   Status: 7
Time: 09/13/2012 17:01:11.0000000   Status: 7
Time: 09/13/2012 17:01:12.0000000   Status: 7
Time: 09/13/2012 17:01:13.0000000   Status: 7
Time: 09/13/2012 17:01:14.0000000   Status: 7
Time: 09/13/2012 17:01:15.0000000   Status: 7
Time: 09/13/2012 17:01:16.0000000   Status: 7
Time: 09/13/2012 17:01:17.0000000   Status: 7
...

N.B.:This output does not demonstrate anything about precision of GPS: This is GPS time, and not UTC time. Then is obvious that GPS says that signal, that it thinks to generate each second, has a infinite precision.

GPS_2

This code evolves GPS_1 (it doesn't use event register), implementing the classic paradigm of communication Producer-Consumer, where producer is the interrupt and consumer is a stand-alone thread. The shared information is the number of PPS that are arrived.

GPS_2 code

Consumer(thread):

```
//get shared address (one time)

memory_loc_key = ftok(".",'M');

  if((id = shmget( memory_loc_key,
    sizeof(long int), 0666 ))<0){/*...*/ }

  shm_ptr = shmat (id,NULL,0);

  if((int)shm_ptr == -1){...}

                         [...]

While(1){

    //use data

  printf("I'm the consumer!: %ld \n", *((long
  int*)shm_ptr));

                    /*wait 0.5 sec*/

}
```

Producer(interrupt):

```
  //each time I need  to get shared

  //address

  memory_loc_key = ftok(".",'M');

if((id = shmget( memory_loc_key,
  sizeof(long int), 0666 ))<0){/*...*/ }

shm_ptr = shmat (id,NULL,0);

if((int)shm_ptr == -1){...}


//increment pps_counter

(*((long int*)shm_ptr))++;
```

GPS_2 output

…
bcIntHandlerRoutine
I'm the consumer!: 2926
I'm the consumer!: 2926
pps_routine: 87:144:244.0059356
bcIntHandlerRoutine
I'm the consumer!: 2927
I'm the consumer!: 2927
pps_routine: 87:144:245.0039618
bcIntHandlerRoutine
I'm the consumer!: 2928
I'm the consumer!: 2928
pps_routine: 87:144:246.0059860
bcIntHandlerRoutine
I'm the consumer!: 2929
I'm the consumer!: 2929
pps_routine: 87:144:247.0030130
…

connection_control

This simple code periodically (about 4 sec) requires packet46 to GPS card, and reads in it information about status of connection. Then it prints a message describing this status.

connection_control output

...
Status: No usable satellites
Status: No usable satellites
Status: No usable satellites
Status: No usable satellites
Status: No usable satellites
Status: No usable satellites
Status: No usable satellites
Status: No usable satellites
Status: No usable satellites
Status: No usable satellites
Status: No usable satellites
Status: No usable satellites
Status: No usable satellites
Status: No usable satellites
Status: No usable satellites
Status: No usable satellites
Status: No usable satellites
Status: No usable satellites
...

This output was taken
when the antenna
was disconnected.

Conclusions

The codes that I've briefly explained are simply sample codes, and they try to show functionalities. It would be simple to modify them, in order to satisfy the various needs.

Next week I will make some studies about FermiLab Database structure