2/16/16

This is part of a minor update of work in progress. The goal is to have it all done before the first P2 hits the street. The next revision of P2Explorer will be posted in a few days. This intro will be revised again as the rest falls into place.

Propeller2Explorer

(preliminaruy) Introduction

What is it?
Propeller2Explorer(P2E) is a Windows development environment for working with a Propeller 2. Currently the P2 exists as a partial FPGA implementation, using Parallax's P123 series of FPGA boards (P2V). P2E is presently set up for use with the latest A9 version, simply because I personally want as much Hub Ram as possible.

P2E could be made to work just as well with the A7 or the DE2-115 from Terasic, simply by moving things around and eliminating some buffers. P2E could also be made to work with a Propeller 1, giving your P1 robot instant access to everything P2E currently supports. This would take some work, but it wouldn't take rocket science to get it going.

Currently P2E is a test bed for the P2V.

What?

Example: within P2E are utilities for getting and sending KinectV2 data over USB serial to the P2V. P2E can also send camera data over the same serial channel. This is fine for testing. We can get data into the P2V for it to work on. BUT if P2E were to be used in an actual design, no doubt we would prefer to have a separate serial line for each device and maybe more than one serial line for each device. P2E does not yet have a simple example of multi-serial support. In an actual design P2Host would be rearranged and modified to meet actual design requirements.

So, P2E and P2Host are good for testing design elements, where in actual in actual practice these elements would be mixed and matched to meet actual design goals.

Why does P2E exist?
P2E began as my answer to my need for instant gratification in the presence of profound levels of uncertainty. I fully expect that almost everything I am doing on the computer side today with P2E will eventually be done by the P2 (all by its lonesome). If and when that does happen, I will still be using P2E as part of a robotics platform. The recent arrival of $99 Windows 10 computers has caused me to rethink what should be part of an affordable robot... and what home built robots might be capable of:)

So, P2E is both as a stop gap/heuristic device/my answer to my own needs and as a long term part of a practical robotics platform.

For fast prototyping, the Propeller 1 is a dream. Designing with the Propeller 2 is going to be like living in Disneyland. There is so much that the P2 is able to do that having a little platform to support it... to work around some of the rough edges... is a little bit of sweetness. P2E eliminates potential road blocks and makes missing information or capacity seem far less important. I am very happy that I

came across Processing.org.

P2E is built using Processing.org's Processing 3x environment. Like all such environment's, Processing 3x isn't perfect, but it is free... that's important:) And it is open sourced... that's important. And it is cross-platformed.

P2E, however, is NOT cross-platformed, because the GUI library, which P2E employs, is only available for Windows and Linux environments.

Most importantly, Processing 3x is trustworthy and fun.

Breaking News

Occasionally I find something that I want to do with the P2v, and I need P2E, but Processing 3x just will not cooperate. Here is a perfect example, which also explains **a bit of a hole in the current user experience:**

I wanted to do stereo-analysis on the P2. I wanted to do it in a way that would be useful to people building robots today. That way, even if I fail in my eventual goal (of building an advanced 3D system to mimic the brain's stereo-vision), the effort won't be a waste of time.

So, I wanted the P2 to take a direct input from a stereo-camera. That's something which can't happen right now... later for sure, but not today. So, I'm using P2E as an intermediary.

Processing 3x has a native interface for USB cameras. I have a USB stereo-camera, the Bloggie3D. The Bloggie3D actually has an available webcam type interface, but it is old and no matter what I tried, Processing 3x could see the Bloggie3D, but it could not access Bloggie3D images.

The work-around was to find a another (free) application that actually could get images from the Bloggie3D. Obvious, right?

ImageJ is an open source image processing application from NIH, which fits the bill. But so far I have not been able to get Processing to talk to ImageJ and I don't want to give ImageJ its own serial link to the P2... too many wires. And I don't want to waste any more time trying.

The current “solution” is to run ImageJ so that the stereo camera view is on the computer screen, and then use a Java robot (software) which takes screenshots of that part of the screen where the Bloggie3D image is located.. This makes it possible for Processing (P2E) to grab a stereo image from the screen and send that image to the P2 any time the P2 wants it or when I prompt P2E from the keyboard. On the P1 you wouldn't be sending a full image and you would have to reverse engineer the analysis but it could work very similarly.

No respectable programmer would ever stoop so low or allow such a monstrosity as this work-around to come out of the closet. A program to write to the screen, another program to capture the screen and then send that along to another program (on the P2) to actually use it?

If they gave out Darwin awards for programming... this is the kind of thing they would award.

No need to get upset about it, it's a kludge to work around a problem that won't be there later.

The really, really ugly part of all this is that Java robots are somewhat alien to Processing 3x... You have to import them from a foreign land... in the form of Java libraries. You don't know which will actually work, which are safe, which will be compatible or which ones will blow up the minute you turn your back on them.

Like all aliens, they are a largely a mystery. While it all works. It just barely works. If you quit out of P2E, you sometimes have to quit out of the entire Processing environment to get P2E to work again... and occasionally you have to repeat this process. Very unbecoming.

Bottom line: occasionally, Java isn't quitting right. I am running all of this on a $99 Win10 Kangaroo. I rarely quit out of P2E except when I am changing it.

That's the bird's eye view of this wormhole:) **If any of this bothers you** and you have no need for a stereo camera... **rip the Java robot out of P3E** until you actually need it... it's only a line or two, which you can easily find by searching in the P2E sketch, for "robot."

However, **ImageJ also supports multiple camera views better than Processing 3x does**... so before you rip out the robot, make sure that you aren't interested in supporting multiple cameras on your P1 or P2 robot.

On to the good stuff.

"What is it good for?"

Basically, everything. And unlike the general theory of relativity, we don't have to wait a hundred years to see if it works.

My long term robot project is a semi-autonomous wheel-chair. I want the chair to have a sense of what is around it... to be able to see. I want the chair to know where it is in the world and what kind of terrain surrounds it. In the first prototype, the chair won't have to recognize very much: the sizes and movement of near objects and the layout of the terrain. Later, the requirements will go up.

To properly solve these kinds of problems, we really should use multiple sensors from different classes. We want to fuse all of the data together, and do it in such a way that critical errors are almost impossible. Every sensor has both capabilities and limitations... the trick is to be able to discern that a sensor is operating within its capabilities or that it is bumping up against the presence of a particular limitation. Generally, we can't do this with a single sensor... we can only do it with multiple sensors. Ideally we want different types of sensors, which share some of the same capabilities but have different limitations. (George Boole was good at this sort of problem:) The more sensors(of different kinds) that we use, the better we should be able to cover the limitations of any particular class and the closer we are to a robust solution.

So, I want every conceivable sensor type/information source that can be reasonably used to see around a wheelchair, to locate the chair in space and to judge the surround in various ways.

Feeding the P2 with a rich stream of data is job #1... making the P2 happy with all of this data is job #2.

P2E is good at feeding the P2. And right now the P2V seems perfectly happy.

P2Explorer is concentrated on sensor types that probably won't be implemented directly for the P2 in the short term. P2Explorer doesn't need to address most of Parallax's inventory, because nearly all of that will become available the instant the P2 becomes available...and probably long before:)

BREAKING NEWS

Last week smart pins v1 became available. A day later, quadrature encoding popped out of the event horizon. PWM is even simpler:)

We don't need to address any of this on the P2E side, except to integrated them into the test bed to test our P2 code base.

This is also true for sensors, such as ultrasound, IR and laser range finding. I mention these because they are all sensors that should be considered as complementary sensors of different types, which will be directly added to the P2 without a need for an intermediary (P2E) and should be part of any robust system. We can confidently place them in our designs, with little need to prototype... any issue can be resolved using a P1. So much information is already available for these sensors that the capabilities and limitations can be easily sorted.

Sensors now supported by P2E ... multiple USB cameras, a 3D camera, and the KinectV2.

Requirements

Hardware/software
Current: P123 A9 and the supporting software(PX and PNut). P2Host, which comes with P2E is the P2 program that interfaces from the P2 to P2E over USB serial.

NTSC-friendly monitor.
Enough information is already available to support VGA as a future option.

Windows 64bit 8&10 work fine. I have no way to test Win7. s
Processing 3.x latest 64 bit version from Processing.org.

Comfort

Get comfortable with Processing 3x first. Look at the examples sketches, down-load some libraries so that you are familiar with the process. It is all very simple, with tremendous documentation and support.

Installation

More complete installation instructions are available elsewhere in this thread and will be described in a step by step in Chapter 1.

It is really very easy until you get to Daniel Shiffman's KinectV2 stuff which is a little more involved and then you follow his instructions... including switching over to a new KinectV2 driver.

Basically, you need to install Processing 3x, some Processing 3x libraries, some Java libraries and the Kinect V2 packages from Microsoft and make sure everything is working first. Processing 3x will tell you what is missing when you try to run the P2E sketch (So, don't worry too much, if you have missed something, Processing 3x will tell you... (and you could simply start out trying to run the P2E sketch and then install libraries as Processing 3x tells you they are missing)

Once your computer is set up to support the KinectV2... then Processing 3.x can help a lot, but you should set your computer up and get KinectV2 going first). You don't have to use a Kinect, but right now you have to have the libraries installed or you will get all sorts of threatening messages.

I know that everything works for an AMD A8 HP and for my cheepo Kangaroo.

That's really it. Pretty minimal.

How it all works

On the P2 side, there is a main Cog. Cog1 running "Hack_Me", which incorporates a looping serial interface.

"Hack_ME" waits for a byte to arrive through the serial RX pin. Once "Hack_ME" consumes a byte, it traverses through a list of meaningful values to see if that byte contains a meaningful value. If it finds a match, it calls the routine for that match. Dog simple.

The matching values and routines are entirely idiosyncratic. They are just my way of building up my own code base, a record of how to do so stuff, that I don't have try to remember... its my little repository. (You can find a list in my previous post, also documented in Hack_Me.)

Hack_Me has one call that is generally useful: "Get_OP".

On the P2E side of things, you type in the letter "o." P2E sends this along to P2Host and Hack_Me calls "Get_OP." Get_OP is used to test simple combinations of instructions.

The results are placed in the LUT. Hack_Me then receives another byte, "r" which means " read and send the contents of your LUT out serial TX.".

The P2 assembly for Get_OP looks like this:

```
'**********************
Get_OP                                  'used for testing code snippets
        mov temp, #253                  ' set the variable, temp, to the value of 253

        wrlut temp, #0                  ' write the value of temp to LUT RAM location 0.
        ret
'**********************
```

The LUT can store up to 512 long(4 byte) values.

Computer side (in P2E)

Processing 3x sketches usually contain a continuous loop function, **draw()** as the "main" routine.

If you create a function, which gets called from inside of **draw(),** the program will call that function repeatedly in the order in which the call occurs within **draw()**; There are some event functions, which occur outside of the **draw()** loop, such as keyboard and serial events.

P2E uses the event driven **keyReleased()**, to parse through a list of keys that might have been signaled by a keyReleased() event and if it finds a match, executes a case statement for that key. This is where you would add your own functions or delete some. For example, within **keyReleased()** you will find case(111)... "111" is the ascii equivalent for the letter "o." When you type "o" on the keyboard, case(111) inside of **keyReleased()** gets executed.

It looks like this:

```
case(111):
myPort.write(111);   --send the letter "o" to P2Host
delay(250);          --wait 250 milliseconds
iscmd=false;         --set a boolean to indicate that what is coming from P2Host is not a command
i=511;               – variable to indicate the number of bytes to expect (-1)
j=-1;                --a counter "nulled" to -1
whichbyte=-1;        --a buffer pointer/counter "nulled" to -1
getlut=true;         – a boolean
myPort.write(114);   – tell P2Host to send entire contents of LUT from Cog0..."r" key equivalent
break;               – end of case
```

This case is doing four things, sending an ascii character, waiting, sending another ascii character to tell the P2V to send Cog1's lut and then the case gets P2E to ready to receive and handle the contents of the LUT. Other routines then receive the data and display it for you. Notice, that there is no handshaking here. The P2Host doesn't tell P2E that Get_OP is done. P2E simply waits 250 milliseconds and assumes that P2V is done and ready for another instruction. The take home here is that Get_OP is expected to last no more than about 20,000,000 P2 clocks. So, this isn't the place to test large looping structures on the P2V. If you want to do that, then change the delay or add some handshaking.

Most of the time a keyReleased() case looks like this:

```
case (49):
myPort.write(49);
break;
```

The long term goal is to have the P2 directing P2E... there is no need for this at the moment, but the capacity exists. Generally, P2E watches for **keyReleased()** and serial events. P2E needs to know what P2Host expects it to do... handle a byte as data... if so, what kind of data and what to do depending upon the type of data that it is.... OR that the next byte is expected to be an instruction coming from P2.

When a P2E function needs data from the P2, P2E does this by setting **iscmd** equal to false, which effectively means that anything coming from the P2 will now be seen as data, not as an instruction.

This is the current serial event handler for P2Explorer

```
void serialEvent(Serial myPort) {
  inByte = myPort.read();
  if (iscmd==true) {
    println("iscmd true");
    switch(inByte) {
      case (1):
      iscmd=false;

      sendimage();
      iscmd=true;
      break;
    }
  }

  if ((getlut==true)&&(iscmd==false)) {
    j=j+1;
    inBytes[j] = inByte;


    if (j>=2047) {
      //how many times has a button been released
      iscmd=true;
      println("ok iscmd is true");
      getlut=false;
    }
  }
}
```

Just to repeat:
Notice that there is only one command implemented... and it matches a byte value of 1.

```
 if (iscmd==true) {
    println("iscmd true");
    switch(inByte) {
      case (1):
      iscmd=false;

      sendimage();
      iscmd=true;
      break;
    }
  }
```

This is a placeholder to ensure the ability of the P2 to take control, but it is not required yet. The normal state for P2E is that **iscmd** is true. If you want it to receive data, your routine has to set **iscmd** to false... otherwise, when P2E sees a byte containing a “1”... it is going to assume that is an

instruction.

When your routine completes, you set **iscmd** back to true before exiting.

Currently P2Host has a primitive mail mechanism to allow signaling between Hack_ME and other cogs. The mail gets cleared in Cog0 as P2Host is initializing itself. Each Cog gets a byte of mail in Hub Ram. Right now it is used as a simple ON/OFF switch. The example for this is Stereo-Point.

That is the basic structure. Pretty simple, huh?

Chapter 1

...later:)