Fun with Charlieplexing

Back in May, my friend Ken Gracey (CEO of Parallax) tweeted about a project they’re working on: a Propeller-powered convention badge for any event that needs one. This was inspired by the success of the DEFCON 22 badge which I had the privilege to help design and write code for. There are lots of tech conventions around the world, yet not everyone has the budget to design custom hardware. This is where Parallax intends to help out. They’ll build the badge — you write cool custom code for your venue.

At the time of this writing, the badge design is not completely finalized, though there are a couple LED circuits that won’t change. Figure 1 is a 3D model of the preliminary design. In one section, there are six blue LEDs connected to three resistors and three I/O pins; in another section, the six LEDs of two RGB modules are similarly wired. Wait ... how do we control six LEDs with only three I/O pins? Charlieplexing!

Charlieplexing — named after its inventor, Charlie Allen — is a method of LED control that takes advantage of the processor’s ability to manipulate the pin state: a pin can float (input mode); be an output and low; or be an output and high (the term tri-state refers to the pin’s ability to be in one of three states). We don’t often think of the input state as an output control mechanism, but it’s critical to the operation of Charlieplexing. Okay, then, let’s start dirt easy with two LEDs (see Figure 2).

If we make P0 high and P1 low, current will flow through D1 and light; D2 is reverse-biased and will not light. If we reverse the outputs and make P0 low and P1 high, D2 will light and D1 will be extinguished.

Now, what if we make P0 or P1 an input (i.e., floating)? When this happens, the pin is disconnected from the output driver which is like opening a switch. Both LEDs will be extinguished, regardless of the state of the other pin. This is key to Charlieplexing.

With this understanding, have a look at Figure 3 and the corresponding logic table in Figure 4. The circuit shows the beauty of the Charlieplex arrangement: With just three I/O pins, we can control up to six LEDs. Getting down to the nitty-gritty, using Charlieplexed connections allows us to control up to \( n + 1 \) LEDs with \( n \) I/O pins. This means four pins could control 12 LEDs, five pins could control 20, and so forth.

As you study the table in Figure 4, you’ll see that one pair of outputs which is connected across the target LED is set to high (H) and low (L), while the third output is set to the input/floating state (X);
This is pretty cool, right? It is, and yet I know you’re thinking, “Okay, Jon, that’s great, but what happens when I want to have two LEDs on at the same time?” As my friend John B. frequently quips ... no problem, it’s just a small matter of programming (SMOP).

So far, we’ve only used Charlieplex wiring, but the software process of Charlieplexing – like its cousin, multiplexing – is more involved. What we’re going to do is loop through a control value that tells us which LEDs are on. For those that are on, we will light them briefly, then move to the next LED.

If we do this fast enough (greater than 50 Hz), the persistence of vision in our retina will cause the LEDs to appear to be on at the same time.

In most processors, this happens in an interrupt, and for very simple Propeller programs we might even be able to use the timer object to refresh the LEDs on a reasonably fixed schedule. To be on the safe side, however, and keep things general-purpose, I decided to create a stand-alone object.

For the object to work, the LEDs must be wired as in Figure 3 and the pin passed to the start() method is the LSB pin of the three-pin group. After that, everything else takes care of itself. Here are the start() and stop() methods for the six LED Charlieplex object:

```
pub start(cp0)

stop

cycleticks := (clkfreq / 300) / 6
cog := cognew(charlie_6(cp0), @stack) + 1
return cog

pub stop

if (cog)
    cogstop(cog - 1)
cog := false

ledbits := @000000
```

It is convention for the start() method to call the stop() method to ensure everything is clean before launching a cog (as we do here). The stop() method will kill the presently running cog, mark it as stopped, and clear the control value (ledbits).

The start() method sets the timing for LED updates in the global (to the object) variable cycleticks. Remember that all timing in the Propeller is done in system ticks. By dividing clkfreq (system ticks in one second) by 300, we set the update rate to 300 Hz. Why? In a world filled with

The method counts on having pin constants CP0 and CP2 defined; these are the LSB (Least Significant Bit) and MSB (Most Significant Bit) control pins of the three-pin group. At the start of the method, we set the direction bits for the group to %000 which disables all outputs. This serves two purposes: 1) It clears the LEDs in the event we pass an invalid LED number, and 2) It prevents ghosting of LEDs when making changes from one to another.

If you look carefully, you’ll see that the code in the case structure directly corresponds with the control table. It’s a good idea to create such a table when using Charlieplex wiring and structure the code to follow your table.

this allows us to control the designated LED. Yes, this means that we can only activate one at a time, and for many applications this is fine. For a 1-of-N project, we don’t need any special code, just a method to handle which LED is active:

```
pub set_cp Led(n)

  dira[CP2..CP0] := %000
  case n
    0: 
      dira[CP2..CP0] := %001
      dira[CP2..CP0] := %011
    1:
      dira[CP2..CP0] := %010
      dira[CP2..CP0] := %011
    2:
      dira[CP2..CP0] := %001
      dira[CP2..CP0] := %101
    3:
      dira[CP2..CP0] := %100
      dira[CP2..CP0] := %101
    4:
      dira[CP2..CP0] := %010
      dira[CP2..CP0] := %110
    5:
      dira[CP2..CP0] := %100
      dira[CP2..CP0] := %110
  esac
```
video cameras, this minimizes odd visual artifacts when things are moving. Note that we further divide the timing by six (the number of LEDs we’re controlling) to ensure that all LEDs are updated at 300 Hz.

The cog.new call for a Spin cog requires two parameters: 1) the method that will run it in its own cog; and 2) the hub address of an array used for the cog’s stack space. As with launching PASM cogs, cog.new will return 1 (cog did not launch) to 7, which gets promoted to 0 (false, did not launch) or 1 to 8 when everything launched normally.

The Charlieplex process is simple enough that we can run it in a Spin cog without reserving to assembly, so long as we keep the application clock frequency at 20 MHz or higher (more on this later). Here’s the method that runs in its own cog:

```spin
pub charlie_6(cp0) l cp2, t, cycle

    cp2 := cp0 + 2
    t := cnt
    repeat
        repeat cycle from 0 to 5
            dira[cp2..cp0] := $000
            if (ledbits & (1 << cycle))
                case cycle
                    0:
                        outa[cp2..cp0] := $001
                        dira[cp2..cp0] := $011
                    1:
                        outa[cp2..cp0] := $010
                        dira[cp2..cp0] := $011
                    2:
                        outa[cp2..cp0] := $001
                        dira[cp2..cp0] := $101
                    3:
                        outa[cp2..cp0] := $100
                        dira[cp2..cp0] := $101
                    4:
                        outa[cp2..cp0] := $010
                        dira[cp2..cp0] := $110

        waitcnt(t += cycleticks)
    end

5:
    outa[cp2..cp0] := $100
    dira[cp2..cp0] := $110

pub set_led(n, state)
    if ((n => 0) and (n <= 5))
        if (state)
            ledbits |= 1 << n
        else
            ledbits &= !(1 << n)

pub set_all(bits)
    ledbits := bits & $111111

pub run_larson(cycles, ms) l n
    repeat cycles
        repeat n from 0 to 4
            leds.set_led(n, true)
            time.pause(ms)
```
when using Charlieplexed LEDs. When using the simple circuit shown in Figure 3, the LEDs should be of the same type and forward voltage. If you’re going to mix LEDs, remove the resistors shown and replace each LED with an appropriate LED/resistor combination.

Another thing that one must watch for is the resistor values. Have a look at Figure 5; what this illustrates is the possibility of lighting multiple LEDs under the right conditions. When we light D1 (P0 high, P1 low, P2 floating), there can be a secondary path through D3 and D6 as shown.

This can happen if the combined forward voltage of the LEDs is lower than the output voltage of the controller, and the resistors are so small that enough current can flow to light both LEDs — though they won’t be as bright as the target LED.

For my circuit, this isn’t a problem because the forward voltage of each LED is about two volts; there is no way to light two of these LEDs in series with a 3.3V micro. That said, if I moved this circuit to a 5V micro, I would have to check the resistor values.

Finally, I mentioned earlier that the Spin driver will run down to a clock speed of 20 MHz (PLL = 4x). We can speed things up by converting the Charlieplex operations to PASM; this will allow us to run all the way down to 5 MHz (PLL = 1x). This might be helpful when using the Charlieplex circuit in a battery powered application.

I’ve included a duplicate of the demo code which uses the PASM version at the article link. Those of you that are starting to make the move from Spin to PASM will find the code interesting. The PASM is very simple, and a near direct match of the Spin code.

**Lock It Down!**

If you’re like me, you have solderless breadboards
with parts in them that never come out. In my case, many of these are in the form of Propeller Activity boards (PABs). My friends at Parallax have introduced a new product that lets me keep my circuit and free up my Activity boards. It's called the Circuit Overlay Board (Figure 6), and it's compatible with the Activity board and the various "Boards of Education" from Parallax. Plus, it's less than six dollars!

You can see the overlay board in action in Figure 7. On the left is an Activity board with the Charplexing circuit installed. On the right is another with the same circuit soldered into an overlay PCB (printed circuit board). Like any shield, I can pop this in and out at my leisure — the PAB breadboard is no longer locked up with a favorite circuit.

To make matters even better, Parallax has released the DipTrace design files for the overlay PCB. The release of these files will facilitate — and I hope encourage — the creation of shields for the Propeller Activity board. I've certainly got plans. Will you join me?

Timer Update

Every day, I seem to find another use for my timer.

Design Your Solution Today

CALL (480) 837-5200

www.embeddedARM.com

TS-7970 Single Board Computer
Industrial High Performance i.MX6 Computer with Wireless Connectivity and Dual GbEth

- 1 GHz Solo or Quad Core Freescale i.MX6 ARM CPU
- 2 GB RAM, 4 GB eMMC Flash
- WiFi and Bluetooth Module
- 2x Gigabit Ethernet, 4x USB
- HDMI, LVDS, & Audio In/Out
- Linux, Android, QNX, Windows

Starting at $169 (QTY 100)
Modulo Starting At $89 (QTY 100)

TS-TPC-7990 Touch Panel PC
7" High End i.MX6 Mountable Panel PC with Dev Tools Such as Debian GNU and QTCreator

- 7 Inch Touch Panel PC Powered by 1 GHz i.MX6 ARM CPU
- Resistive and Capacitive Screens
- 10 Inch Screen Available
- Linux, Android, QNX, & Windows
- QTCreator, GTK, DirectFB, and More
- Yocto, Debian, Ubuntu Distro Support

Embellished TPCs Also Available
Starting at $299 (QTY 100)

$342 (QTY 1)

We’ve never discontinued a product in 30 years
Support every step of the way with open source vision
Unique embedded solutions add value for our customers