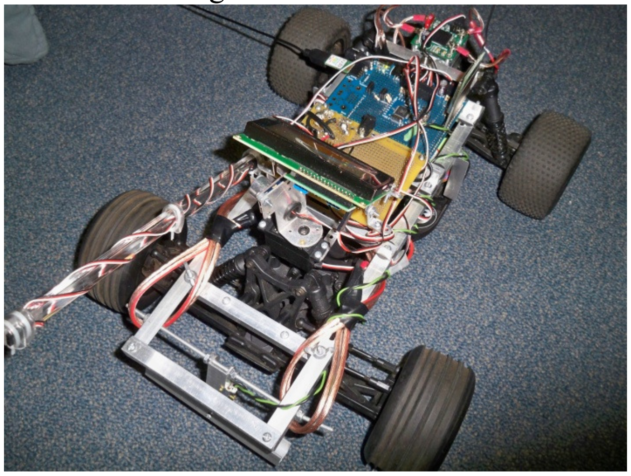
A Propeller Multimode Debugging Tool

SRLM
25 May 2009
SRLM_productions@yahoo.com

Introduction

The Parallax Propeller is a multicore microcontroller with potential to be used in a wide variety of applications. The Propeller has eight identical cores, called cogs, along with a central coordinator called the hub. The Propeller is programmed in a high level language pseudo-object oriented language called Spin and in low level called Propeller Assembly, or PASM. The microcontroller has 32 I/O, a clock speed of up to 80MHz (with provisions for overclocking to 96MHz), and relatively few hardware features. The most notable hardware features are circuits for video generation (both AV and VGA formats). There are no serial communication circuits, no ADCs, and no interrupts. These functions are instead done in software, utilizing the unique multi-cored design.


A robotic platform tethered by a serial cable.

The Propeller relies on the programmer to dedicate individual cogs, and to select the correct library code for the application requirements. There are no provisions for compile-time optimization or division of code. Instead, the system is entirely deterministic: the programmer is in charge of what happens when.

This paper presents one of the problems that comes up with robotics development, and proposes a solution for platforms that utilize the Propeller. This solution is then field tested and proven to be beneficial.

One of the constant problems that robotics developers face is a lack of information when the robot is being tested and is not connected via a serial cable. Since most debugging data flows through this cable many times the roboticist is left at ends to determine the cause of failure when the robot eventually crashes. A better solution would be to send data wirelessly through a radio modem. Unfortunately, to do so would require that the researcher always have a receiver available, and this is not always the case.

Presented here is a solution to this dilemma: using the power of the Propeller, a debugging solution is proposed and implemented that utilizes three methods of sending the data to the user: via a serial cable, via a wireless link, and via an onboard LCD the robot itself. These three devices are seamlessly integrated into a single function call eliminating the hassle of changing code based on the required output destination.

A description of the use

The software presented here is a multimode wrapper (terminal, wireless, and LCD) around the four port serial driver available in [1]. It is completely compatible with the full version of the driver, but for this wrapper the driver has been edited down to save hub RAM.

To begin using this software all that is needed is to start the wrapper (by calling the start function with the appropriate parameters). From there, it follows the basic format familiar to most serial drivers.

For example, to send a single byte the user can type something similar to as follows:

```
CON
a = 97      'The ASCII value for the letter a
OBJ
debug : "DebugMultimode"
PUB Main
...
debug.tx(a) 'Will output the ASCII letter a
```

To output a whole string, the user can code something like this:

```
OBJ
debug : "DebugMultimode"
PUB Main
...
debug.str(string("Hello World!"))
```

In addition to strings and bytes, the wrapper allows the user to output a decimal number (unpadded). Also available are functions that allow the user to output binary and hexadecimal numbers to the specified number of digits:

```
CON
a = 97
OBJ
debug : "DebugMultimode"
PUB Main
...
debug.bin(a, 8)
debug.dec(a)
debug.hex(a, 2)
```

A description of the wrapper operation

The wrapper has several key features that should be considered. First, the LCD screen specified in the argument to the start function has a limited number of display characters (unlike the effective screen of a terminal or wireless link), so the user should be careful of what is output. The LCD screen can handle only a finite number of characters and update speed is relatively slow, so debug data must be chosen carefully.

The LCD operation is special relative to the operation of the terminal link and wireless link. The wrapper has special code to handle wrap-around on the LCD screen when characters overflow the last row. For example, on a 4x20 LCD screen (4 rows, 20 columns) if the user debugs 81 characters then the first 80 are erased from the screen, and the last character is output in the first location (row 0, column 0). Secondly, a CR input to the wrapper must come from either a call to the `cr` function or the `tx` function. If the CR is embedded in a string then the LCD will not display data properly; the entire wrapper must be stopped and restarted.

The wrapper always transmits on all three channels, and is not selective. This allows the user to not have to worry about what data appears where: by using this wrapper, they simply make the call once in their program, and the data is sent out on all possible channels. This avoids placing the burden on the user of deciding where to send debug information: the data is sent out to all channels.

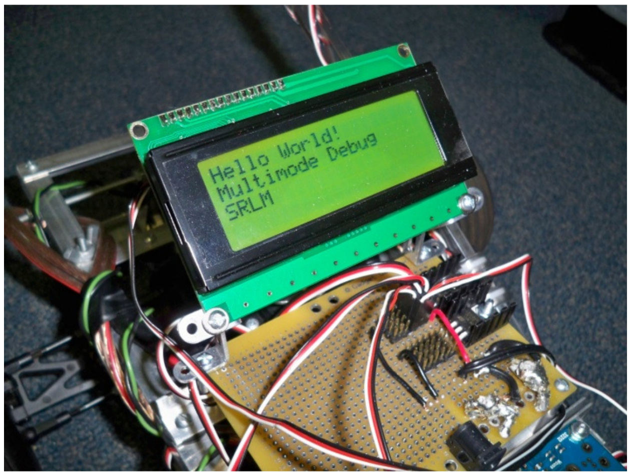
The final consideration to consider is the wireless link hardware. The best radio modem would have a large buffer for both transmitting and receiving. This allows large amounts of data to be communicated efficiently. Care should be taken to minimize the amount of data if there is no buffer.

Finally, the full wrapper and driver consume roughly 800 longs of space and use a single cog.

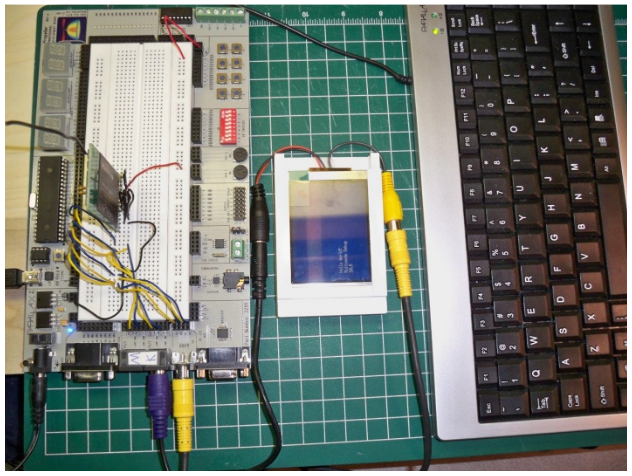
A full test setup

This section describes the steps required to test a full setup.

Requirements:
Propeller A (slave): 4x20 LCD, RF transceiver, terminal connection
Propeller B(master): RF transceiver, keyboard, tv



The serial LCD displaying received data on the slave device.

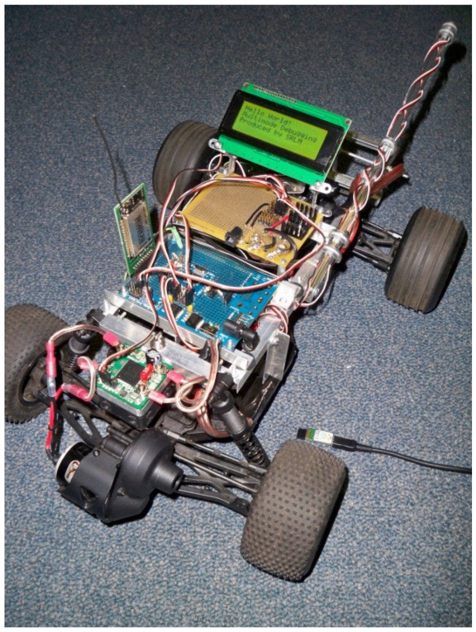


The master Propeller, components from left to right: Propeller professional development board with mounted Propeller and wireless transceiver; tv screen displaying received data; keyboard.

Load the appropriate programs (included in the zipped code file) into the correct Propeller after making sure that pin definitions match. Turn on both, with the slave connected to a computer terminal via a prop-plug. You should see what you type appearing at each location: tv, terminal, and LCD. If successful, this indicates that the slave Propeller has received the data from the wireless link, retransmitted it on all three channels, and that the master has received the character over the wireless link.

Conclusion

The Propeller is a capable multi-cored microcontroller whose unique architecture provides new possibilities over an interrupt based system. By using a cog as a debugging function, the multimode wrapper overcomes one of the challenges of developing mobile robotics: the debug data must go to different destinations depending on what type of test is being done. If the robot is tethered by a USB cable, then it is useful to have a terminal on the computer to receive data. If the robot is maneuvering on a course and being followed by a researcher it is useful to have the debug data appear on the robot itself. If the researcher is at their computer while the robot maneuvers, it is useful to have the debug data sent wirelessly to a base station. The multimode wrapper takes care of all these cases, and allows the programmer to use a single debug statement in their code that is then branched to all three sources.


The robot no longer tethered by a cable

Resources

[1] T. Moore, Author, *Multiple Serial Port Driver: Version 1.1*. [Online] Parallax: 2008. Available: http://obex.parallax.com/objects/340/ [Accessed: 25 May 2009]