BlinkyButton PRU Project

(version 0.1-04.28.2019) Takis Zourntos (takis.zourntos@emads.org)

The BlinkyButton PRU Project is intended to provide a learning context for developing applications with the real-time hardware resources of the AM335x MCU, under Linux. The AM335x is the basis for the open-source Beaglebone Black and Pocket Beagle development platforms and is a powerful chip for embedded applications, featuring 32-bit-wide data and address buses, 1-GHz clock frequency, ARM A8 core and a variety of useful microcontroller-esque resources like PWM, A/D conversion, flash-memory interfacing, as well as the usual peripheral buses (I2C, SPI, UART, CAN, etc.). In addition, the AM335x features two on-chip programmable real-time units (PRUs) which are themselves 32-bit RISC cores operating at 200-MHz. Because real-time operation is often critical in embedded systems applications, getting into the PRU subsystem (PRUSS) is important for developers who like the potential of Linux but need a hard real-time solution. This project covers the full range of a PRU application, from electronics to software. The full project is available here, a Git repository.

PRU Overview

As mentioned, there are two PRUs in the Beaglebone Black (BB) or Pocket Beagle (PB) AM335x MCU, denoted as PRU0 and PRU1. Each has a set of enhanced GPIO pins (which can operate at a much higher frequency than conventional GPIOs) designated as follows:

```
pr1_prX_pru_r3Y_Z
```

in which:

X is the PRU number (0 or 1)
Y defines whether the pin in an input or output (1=1nput, 0=0utput)
Z is the pin number (0 through 16, for a total of 17 pins)

For example, `pr1_pru0_pru_r30_6`, denotes the 7th gpio pin for PRU0 which is an output. Consult Derek Molloy's excellent P8/P9 (P1/P2 on the PB) header charts for all available enhanced GPIO pins (available via Mode 5 or Mode 6 muxing), which can be found here. Essentially, register r30 (of either PRU0 or PRU1) has a bit-for-bit correspondence with an enhanced GPIO output, and, similarly, register r31 has a 1:1 bit-gpio relationship with the enhanced GPIO inputs. Please note that not all PRU pins are available via the headers, and you should disable hdmi video in *uEnv.txt* for full access to the enhanced GPIOs. For further reading, please see Chapter 15 of Derek Molloy's comprehensive *Exploring Beaglebone*, Second Edition. Also, please be aware that things may have changed substantially in the Beaglebone Universe since the time of this writing: April 29th, 2019.

Note the following *uEnv.txt* file boot options. Choose the PRU Overlay that matches your version of the kernel (`$ uname -a`):

```
###PRUSS OPTIONS
###pru_rproc (4.4.x-ti kernel)
#uboot_overlay_pru=/lib/firmware/AM335X-PRU-RPROC-4-4-TI-00A0.dtbo
###pru_rproc (4.14.x-ti kernel)
uboot_overlay_pru=/lib/firmware/AM335X-PRU-RPROC-4-14-TI-00A0.dtbo
###pru_uio (4.4.x-ti, 4.14.x-ti & mainline/bone kernel)
#uboot_overlay_pru=/lib/firmware/AM335X-PRU-UIO-00A0.dtbo
###
```

We do both UIO and `remoteproc` versions of this project. While UIO is being phased out (meaning, despite the availability of a UIO option in *uEnv.txt*, there may be kernel incompatibilities with UIO that prevent us from using it), conceptually I like the idea of having a Linux "host program" to set up the system before the PRUs start running. On the other hand, `remoteproc` seems more streamlined and allows for some control over the PRUs via filesystem commands. Note that there are some differences in the assemblers available for UIO and `remoteproc`, in terms of notation and in how the instruction set is interpreted. For example, with *pasm*, it is acceptable to use the MOV instruction to place an immediate value into a register. However, this is not allowed with the `remoteproc` assembler, and you must use the LDI32 instruction. Also, with load and store instructions (like LBBO and SBBO), you must precede the first register argument with an ampersand symbol, "&". And the CLR and SET syntax under pasm required only one argument, but with the new assembler, you must use two (the first of which seems redundant!). Minor differences to be sure, but they can cause a developer frustration when porting from UIO to `remoteproc`.

To configure the BB or PB pads/pins, you can use (or compose a shell script that makes use of) the **config-pin** command. For example, the **config-pin** command can be used on a specific pin as follows:

```
debian@beaglebone:~$ config-pin -l p1_29
default gpio gpio_pu gpio_pd gpio_input qep pruout pruin
debian@beaglebone:~$ config-pin p1_29 pruout
debian@beaglebone:~$ config-pin -q p1_29
P1_29 Mode: pruout
```

A shell script, *pruproj_startup.sh*, which configures the pins automatically on the PB, is provided. Of course, it must be executed before the PRUs run and produces the following output:

```
debian@beaglebone:~$ sudo ./pruproj_startup.sh
Configuring eGPIO pins for PRUSS blinky-button project

current pin status
P1_29 Mode: default Direction: in Value: 0
P1_31 Mode: default Direction: in Value: 0
```

```
P1_30 Mode: default Direction: in Value: 0
P1_32 Mode: default Direction: in Value: 0

updated pin statuses
P1_29 Mode: pruout
P1_31 Mode: pruout
P1_30 Mode: pruin
P1_32 Mode: pruin
```

Note that for this application, we are using the Pocket Beagle and the following pins:

pin	PRU register and bit	note
egpioLEDred="p1_29"	pr1_pru0_pru_r30_7	PRU0, output
egpioLEDblue="p1_31"	pr1_pru0_pru_r30_4	PRU0, output
egpioButton1="p1_30"	pr1_pru1_pru_r31_15	PRU1, input
egpioButton2="p1_32"	pr1_pru1_pru_r31_14	PRU1, input

Interestingly, p1_30 and p1_32 have no counterparts on the BB headers. So if you are adapting this project to the Beaglebone, you will need to select two PRU1 enhanced GPIOs pins from those available on that platform. The pins were chosen as above because they are all in close proximity on the PB header.

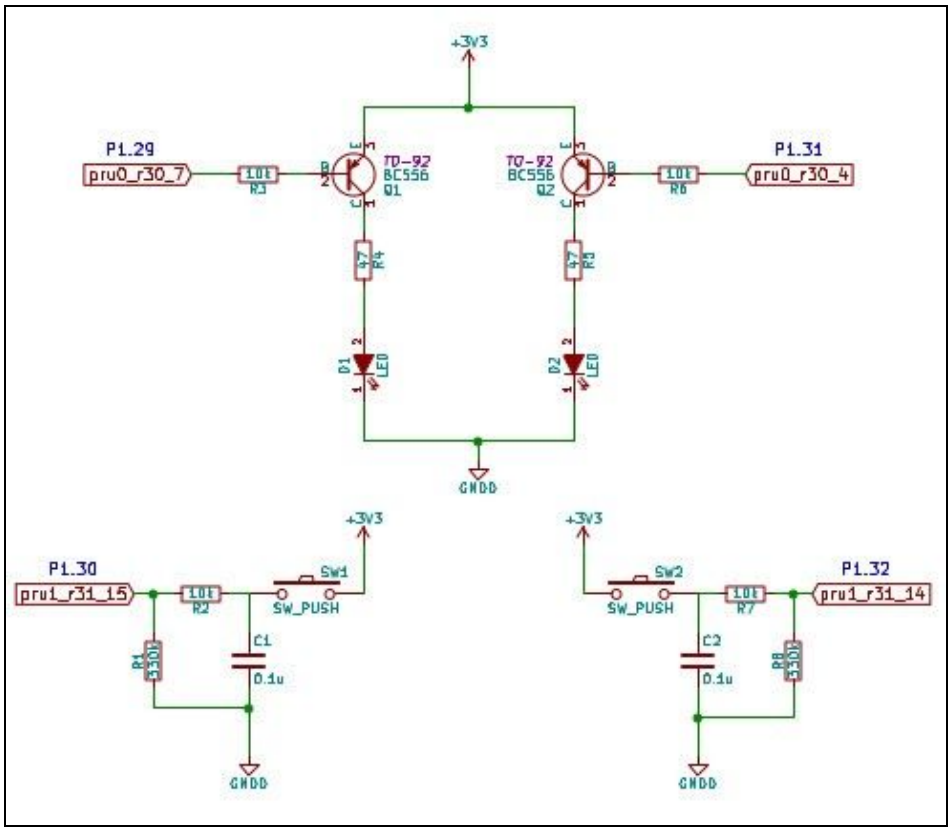

Figure 1: *BlinkyButton PRU Project on PB Schematic*: the BC556 PNP transistors are general-purpose

small signal transistors (you can use your favourite substitute, like the 2N3906, for example)

Project Description

This project uses a simple electronic circuit to illustrate the functioning of the PRUs and enhanced GPIOs. In essence, the circuit consists of two alternately flashing LEDs and two push buttons, along with transistors to drive the LEDs (so that the delicate pads of the AM335x are not overloaded) and various passive components. Note that there are no direct connections between the LEDs and buttons. The idea is to have button control of the LEDs mediated by the MCU. Pressing Button1 decreases the flash frequency, while pressing Button2 increases the flash frequency, as illustrated in this video of a working project. Pressing both buttons at the same time should exit the application [this part is still buggy, as in, at the moment, the application does not halt by itself--- working on that and apologies]. By the way, the system shown uses a green LED rather than a blue one, which does not match with my labelling conventions. Further apologies.

The interesting aspect of the project is that both PRUs are used at the same time. PRU0 handles the LED blinking, and PRU1 handles the buttons and updates the delay variable. Furthermore, a combination of *C* and assembly language programming is used. The circuit schematic is shown in Figure 1.