

PRU Examples

Getting Example Code Problem

You are ready to start playing with the examples and need to find the code.

Solution

You can find the code (and the whole book) on the PRU Cookbook github site: <https://github.com/MarkAYoder/PRUCookbook/tree/master/docs>. Just clone it on your Beagle and then look in the **docs** directory.

```
bone$ git clone https://github.com/MarkAYoder/PRUCookbook.git
bone$ cd PRUCookbook/docs/
bone$ ls -F
01case/      05blocks/  book.html      header.adoc    notes.adoc
02start/     06io/      book.pdf       index.adoc     notes.html
03details/   07more/    copyright.adoc index.html     style.adoc
04debug/     book.adoc  hack.sh*       Makefile      style.html
```

Each chapter has its own directory and within that directory is a **code** directory that has all of the code.

```
bone$ cd 02start/code/
bone$ ls
AM335x_PRU.cmd  hello.c  Makefile  resource_table_empty.h
setup.sh
```

Go and explore.

Blinking an LED Problem

You want to make sure everything is set up by blinking an LED.

Solution

The 'hello, world' of the embedded world is to flash an LED. [hello.pru0.c](#) is some code that blinks the `USR3` LED ten times using the PRU.

hello.pru0.c

```

1  #include <stdint.h>
2  #include <pru_cfg.h>
3  #include "resource_table_empty.h"
4  #include "prugpio.h"
5
6  volatile register unsigned int R30;
7  volatile register unsigned int R31;
8
9  void main(void) {
10     int i;
11
12     uint32_t gpio1 = (uint32_t *)GPIO1;
13
14     / Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
15     CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;
16
17     for(i=0; i<10; i++) {
18         gpio1[GPIO_SETDATAOUT] = USR3;          // The the USR3 LED on
19
20         delay_cycles(500000000/5);    // Wait 1/2 second
21
22         gpio1[GPIO_CLEARDATAOUT] = USR3;
23
24         delay_cycles(500000000/5);
25
26     }
27     __halt();
28 }
29
30 // Turns off triggers
31 #pragma DATA_SECTION(init_pins, ".init_pins")
32 #pragma RETAIN(init_pins)
33 const char init_pins[] =
34     "/sys/class/leds/beaglebone:green:usr3/trigger\0none\0" \
35     "\0\0";

```

Later chapters will go into details of how this code works, but if you want to run it right now do the following.

Running Code

```

bone$ git clone https://github.com/MarkAYoder/PRUCookbook.git
bone$ cd PRUCookbook/docs/02start/code
bone$ source setup.sh
TARGET=hello.pru0
bone$ make
/var/lib/cloud9/common/Makefile:28:
MODEL=TI_AM335x_BeagleBone_Black, TARGET=hello.pru0, COMMON=/var/lib/cloud9/common
/var/lib/cloud9/common/Makefile:147: GEN_DIR=/tmp/cloud9-examples, CHIP=am335x, PROC=pru, PRUN=0, PRU_DIR=/sys/class/remoteproc/remoteproc1, EXE=.out
- Stopping PRU 0
- copying firmware file /tmp/cloud9-examples/hello.pru0.out to /lib/firmware/am335x-pru0-fw

```

```
write_init_pins.sh
writing "none" to
"/sys/class/leds/beaglebone:green:usr3/trigger"
- Starting PRU 0
MODEL    = TI_AM335x_BeagleBone_Black
PROC     = pru
PRUN     = 0
PRU_DIR  = /sys/class/remoteproc/remoteproc1
```

Look quickly and you will see the `USR3` LED blinking.

UART Problem

I'd like to use something like `printf()` to debug my code.

Solution

One simple, yet effective approach to 'printing' from the PRU is an idea taken from the Arduino playbook; use the UART (serial port) to output debug information. The PRU has it's own UART that can send characters to a serial port.

You'll need a **3.3V FTDI cable** to go between your Beagle and the USB port on your host computer as shown in [FTDI cable](#). [FTDI images are from the BeagleBone Cookbook <http://shop.oreilly.com/product/0636920033899.do>] You can get such a cable from places such as [Sparkfun](#) or [Adafruit](#).



Figure 25. FTDI cable

Discussion

The Beagle side of the FTDI cable has a small triangle on it as shown in [FTDI connector](#).

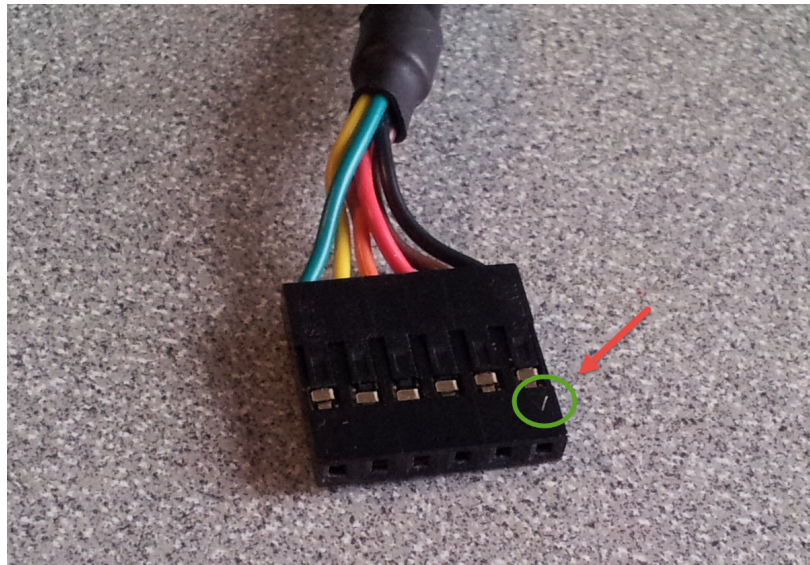


Figure 26. FTDI connector

The connector attaches to the Black on the FTDI pins, shown in [FTDI pins for the FTDI connector](#), with the triangle connecting near pin P9_20 which is the right side of the connector as viewed in the figure.

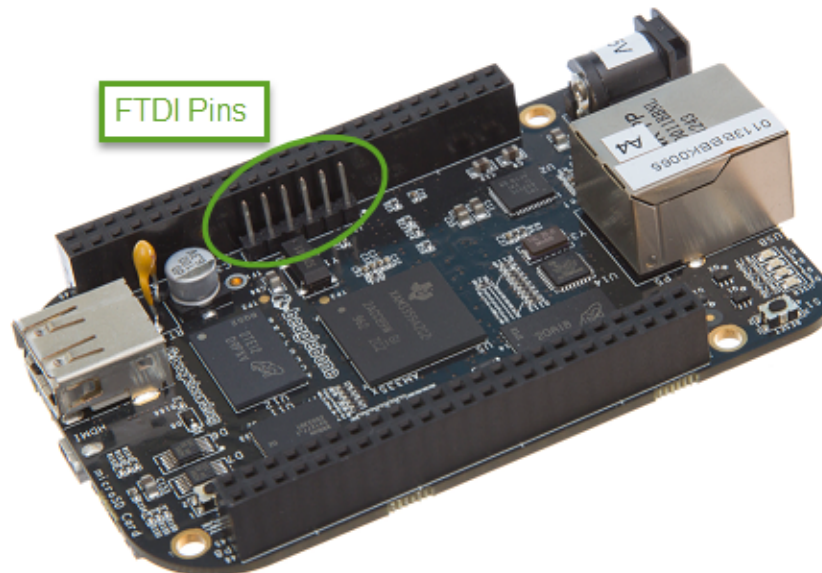


Figure 27. FTDI pins for the FTDI connector

Two examples of using the UART are presented here. The first ([uart1.c](#)) sends a character out the serial port then waits for a character to come in. Once the new character arrives another character is output.

The second example ([uart2.c](#)) prints out a string and then waits for characters to arrive. Once an ENTER appears the string is sent back.

For either of these you will need to set the pin muxes.

config-pin

```
# Configure tx
bone$ config-pin P9_24 pru_uart
# Configure rx
bone$ config-pin P9_26 pru_uart
```

uart1.c

Set the following variables so `make` will know what to compile.

make

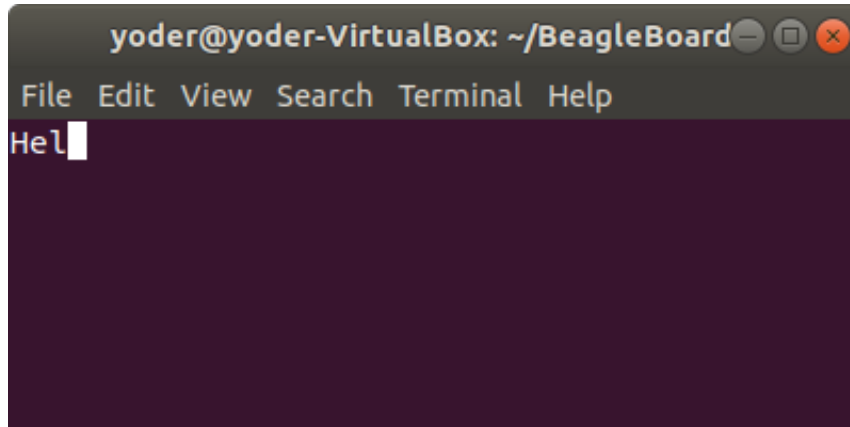
```
bone$ export PRUN=0
bone$ export TARGET=uart1
bone$ make
- Stopping PRU 0
[sudo] password for debian:
stop
CC uart1.c
"uart1.c", line 87: warning #112-D: statement is unreachable
"uart1.c", line 15: warning #552-D: variable "rx" was set but
never used
LD /tmp/pru0-gen/uart1.obj
- copying firmware file /tmp/pru0-gen/uart1.out to
/lib/firmware/am335x-pru0-fw
- Starting PRU 0
start
```

Now `make` will compile, load PRU0 and start it. In a terminal window on your host computer run

```
host screen /dev/ttyUSB0 115200
```

It will initially display the first characters (H) and then as you enter characters on the keyboard, the rest of the message will appear.

uart1.c output



Here's the code (`uart1.c`) that does it.

uart1.c

```
1  // From: http://git.ti.com/pru-software-support-package/pru-software-support-  
2  package/trees/master/examples/am335x/PRU\_Hardware\_UART  
3  
4  #include <stdint.h>  
5  #include <pru_uart.h>  
6  #include "resource_table_empty.h"  
7  
8  /* The FIFO size on the PRU UART is 16 bytes; however, we are (arbitrarily)  
9   * only going to send 8 at a time */  
10 #define FIFO_SIZE 16  
11 #define MAX_CHARS 8  
12  
13 void main(void)  
14 {  
15     uint8_t tx;  
16     uint8_t rx;  
17     uint8_t cnt;  
18  
19     /* hostBuffer points to the string to be printed */  
20     char* hostBuffer;  
21  
22     /*** INITIALIZATION ***/  
23  
24     /* Set up UART to function at 115200 baud - DLL divisor is 104 at 16x oversample  
25      * 192MHz / 104 / 16 = ~115200 */  
26     CT_UART.DLL = 104;  
27     CT_UART.DLH = 0;  
28     CT_UART.MDR = 0x0;  
29  
30     /* Enable Interrupts in UART module. This allows the main thread to poll for  
31      * Receive Data Available and Transmit Holding Register Empty */  
32     CT_UART.IER = 0x7;  
33  
34     /* If FIFOs are to be used, select desired trigger level and enable  
35      * FIFOs by writing to FCR. FIFOEN bit in FCR must be set first before  
36      * other bits are configured */  
37     /* Enable FIFOs for now at 1-byte, and flush them */  
38     CT_UART.FCR = (0x8) | (0x4) | (0x2) | (0x1);  
39     //CT_UART.FCR = (0x80) | (0x4) | (0x2) | (0x01); // 8-byte RX FIFO trigger
```



```

40
41     /* Choose desired protocol settings by writing to LCR */
42     /* 8-bit word, 1 stop bit, no parity, no break control and no divisor latch */
43     CT_UART.LCR = 3;
44
45     /* Enable loopback for test */
46     CT_UART.MCR = 0x00;
47
48     /* Choose desired response to emulation suspend events by configuring
49      * FREE bit and enable UART by setting UTRST and URRST in PWREMU_MGMT */
50     /* Allow UART to run free, enable UART TX/RX */
51     CT_UART.PWREMU_MGMT = 0x6001;
52
53     /*** END INITIALIZATION ***/
54
55     /* Priming the 'hostbuffer' with a message */
56     hostBuffer = "Hello! This is a long string\r\n";
57
58     /*** SEND SOME DATA ***/
59
60     /* Let's send/receive some dummy data */
61     while(1) {
62         cnt = 0;
63         while(1) {
64             /* Load character, ensure it is not string termination */
65             if ((tx = hostBuffer[cnt]) == '\0')
66                 break;
67             cnt++;
68             CT_UART.THR = tx;
69
70             /* Because we are doing loopback, wait until LSR.DR == 1
71              * indicating there is data in the RX FIFO */
72             while ((CT_UART.LSR & 0x1) == 0x0);
73
74             /* Read the value from RBR */
75             rx = CT_UART.RBR;
76
77             /* Wait for TX FIFO to be empty */
78             while (!(CT_UART.FCR & 0x2) == 0x2));
79         }
80     }
81
82     /*** DONE SENDING DATA ***/
83
84     /* Disable UART before halting */
85     CT_UART.PWREMU_MGMT = 0x0;
86
87     /* Halt PRU core */
88     __halt();
}

```

The first part of the code initializes the UART. Then the line `CT_UART.THR = tx;` takes a character in `tx` and sends it to the transmit buffer on the UART. Think of this as the UART version of the `printf()`.

Later the line `while (!(CT_UART.FCR & 0x2) == 0x2);` waits for the transmit FIFO to be empty. This makes sure later characters won't overwrite the buffer before they can be

sent. The downside is, this will cause your code to wait on the buffer and it might miss an important real-time event.

The line `while ((CT_UART.LSR & 0x1) == 0x0);` waits for an input from the UART (possibly missing something) and `rx = CT_UART.RBR;` reads from the receive register on the UART.

These simple lines should be enough to place in your code to print out debugging information.

uart2.c

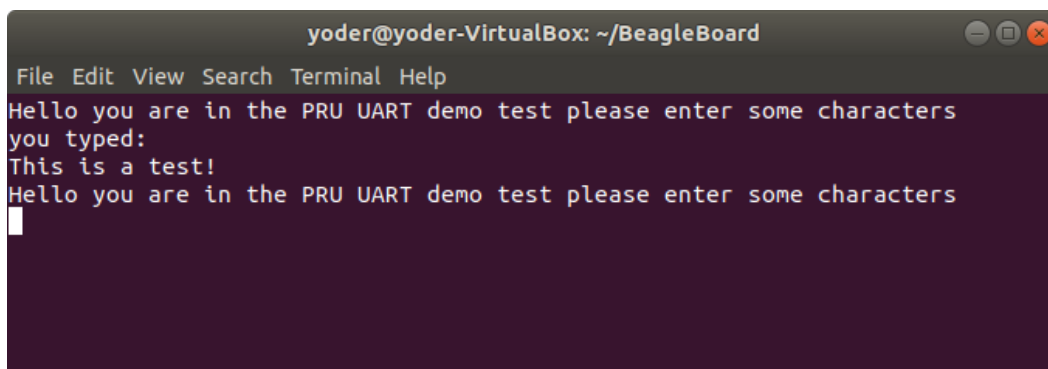
If you want to try `uart2.c`, run the following:

make

```
bone$ export PRUN=0
bone$ export TARGET=uart2
bone$ make
- Stopping PRU 0
stop
CC uart2.c
"uart2.c", line 122: warning #112-D: statement is unreachable
LD /tmp/pru0-gen/uart2.obj
- copying firmware file /tmp/pru0-gen/uart2.out to
/lib/firmware/am335x-pru0-fw
- Starting PRU 0
start
```

You will see:

uart2.c output

A screenshot of a terminal window titled "yoder@yoder-VirtualBox: ~/BeagleBoard". The terminal shows the output of the program: "Hello you are in the PRU UART demo test please enter some characters", followed by "you typed:", then "This is a test!", and finally "Hello you are in the PRU UART demo test please enter some characters" again. A cursor is visible at the end of the last line.

```
yoder@yoder-VirtualBox: ~/BeagleBoard
File Edit View Search Terminal Help
Hello you are in the PRU UART demo test please enter some characters
you typed:
This is a test!
Hello you are in the PRU UART demo test please enter some characters
█
```

Type a few characters and hit ENTER. The PRU will playback what you typed, but it won't echo it as you type.

uart2.c defines PrintMessageOut() which is passed a string that is sent to the UART. It takes advantage of the eight character FIFO on the UART. Be careful using it because it also uses while (!CT_UART.LSR_bit.TEMT); to wait for the FIFO to empty, which may cause your code to miss something.

[uart2.c](#) is the code that does it.

uart2.c

```
1 // From: http://git.ti.com/pru-software-support-package/pru-software-support-
2 package/trees/master/pru_cape/pru_fw/PRU_Hardware_UART
3
4 #include <stdint.h>
5 #include <pru_uart.h>
6 #include "resource_table_empty.h"
7
8 /* The FIFO size on the PRU UART is 16 bytes; however, we are (arbitrarily)
9  * only going to send 8 at a time */
10 #define FIFO_SIZE 16
11 #define MAX_CHARS 8
12 #define BUFFER 40
13
14 /*******
15 // Print Message Out
16 // This function take in a string literal of any size and then fill the
17 // TX FIFO when it's empty and waits until there is info in the RX FIFO
18 // before returning.
19 /*******
20 void PrintMessageOut(volatile char* Message)
21 {
22     uint8_t cnt, index = 0;
23
24     while (1) {
25         cnt = 0;
26
27         /* Wait until the TX FIFO and the TX SR are completely empty */
28         while (!CT_UART.LSR_bit.TEMT);
29
30         while (Message[index] != NULL && cnt < MAX_CHARS) {
31             CT_UART.THR = Message[index];
32             index++;
33             cnt++;
34         }
35         if (Message[index] == NULL)
36             break;
37     }
38
39     /* Wait until the TX FIFO and the TX SR are completely empty */
40     while (!CT_UART.LSR_bit.TEMT);
41
42 }
43
44 /*******
45 // IEP Timer Config
46 // This function waits until there is info in the RX FIFO and then returns
47 // the first character entered.
48 /*******
49 char ReadMessageIn(void)
```

```

50 {
51     while (!CT_UART.LSR_bit.DR);
52
53     return CT_UART.RBR_bit.DATA;
54 }
55
56 void main(void)
57 {
58     uint32_t i;
59     volatile uint32_t not_done = 1;
60
61     char rxBuffer[BUFFER];
62     rxBuffer[BUFFER-1] = NULL; // null terminate the string
63
64     /*** INITIALIZATION ***/
65
66     /* Set up UART to function at 115200 baud - DLL divisor is 104 at 16x oversample
67      * 192MHz / 104 / 16 = ~115200 */
68     CT_UART.DLL = 104;
69     CT_UART.DLH = 0;
70     CT_UART.MDR_bit.OSM_SEL = 0x0;
71
72     /* Enable Interrupts in UART module. This allows the main thread to poll for
73      * Receive Data Available and Transmit Holding Register Empty */
74     CT_UART.IER = 0x7;
75
76     /* If FIFOs are to be used, select desired trigger level and enable
77      * FIFOs by writing to FCR. FIFOEN bit in FCR must be set first before
78      * other bits are configured */
79     /* Enable FIFOs for now at 1-byte, and flush them */
80     CT_UART.FCR = (0x80) | (0x8) | (0x4) | (0x2) | (0x01); // 8-byte RX FIFO trigger
81
82     /* Choose desired protocol settings by writing to LCR */
83     /* 8-bit word, 1 stop bit, no parity, no break control and no divisor latch */
84     CT_UART.LCR = 3;
85
86     /* If flow control is desired write appropriate values to MCR. */
87     /* No flow control for now, but enable loopback for test */
88     CT_UART.MCR = 0x00;
89
90     /* Choose desired response to emulation suspend events by configuring
91      * FREE bit and enable UART by setting UTRST and URRST in PWREMU_MGMT */
92     /* Allow UART to run free, enable UART TX/RX */
93     CT_UART.PWREMU_MGMT_bit.FREE = 0x1;
94     CT_UART.PWREMU_MGMT_bit.URRST = 0x1;
95     CT_UART.PWREMU_MGMT_bit.UTRST = 0x1;
96
97     /* Turn off RTS and CTS functionality */
98     CT_UART.MCR_bit.AFE = 0x0;
99     CT_UART.MCR_bit.RTS = 0x0;
100
101     /*** END INITIALIZATION ***/
102
103     while(1) {
104         /* Print out greeting message */
105         PrintMessageOut("Hello you are in the PRU UART demo test please enter some
106 characters\r\n");
107
108         /* Read in characters from user, then echo them back out */
109         for (i = 0; i < BUFFER-1 ; i++) {

```

```

110         rxBuffer[i] = ReadMessageIn();
111         if(rxBuffer[i] == '\r') {    // Quit early if ENTER is hit.
112             rxBuffer[i+1] = NULL;
113             break;
114         }
115     }
116
117     PrintMessageOut("you typed:\r\n");
118     PrintMessageOut(rxBuffer);
119     PrintMessageOut("\r\n");
120 }
121
122 /** DONE SENDING DATA */
123 /* Disable UART before halting */
124 CT_UART.PWREMU_MGMT = 0x0;
125
126 /* Halt PRU core */
127 __halt();
128 }

```

More complex examples can be built using the principles shown in these examples.

PWM Generator

One of the simplest things a PRU can do is generate a simple signal starting with a single channel PWM that has a fixed frequency and duty cycle and ending with a multi channel PWM that the ARM can change the frequency and duty cycle on the fly.

Problem

I want to generate a PWM signal that has a fixed frequency and duty cycle.

Solution

The solution is fairly easy, but be sure to check the **Discussion** section for details on making it work.

[pwm1.pru0.c](#) shows the code.

pwm1.pru0.c

```
1  #include <stdint.h>
2  #include <pru_cfg.h>
3  #include "resource_table_empty.h"
4  #include "prugpio.h"
5
6  volatile register uint32_t __R30;
7  volatile register uint32_t __R31;
8
9  void main(void)
10 {
11     uint32_t gpio = P9_31; // Select which pin to toggle.;
12
13     /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
14     CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;
15
16     while(1) {
17         __R30 |= gpio; // Set the GPIO pin to 1
18         __delay_cycles(100000000);
19         __R30 &= ~gpio; // Clear the GPIO pin
20         __delay_cycles(100000000);
21     }
22 }
```

- TODO - Add AI

To run this code you need to configure the pin muxes to output the PRU. If you are on the Black run

```
bone$ config-pin P9_31 prout
```

On the Pocket run

```
bone$ config-pin P1_36 prout
```

Then, tell Makefile which PRU you are compiling for and what your target file is

```
bone$ export TARGET=pwm1.pru0
```

Now you are ready to compile

```
bone$ make
/var/lib/cloud9/common/Makefile:29:
MODEL=TI_AM335x_BeagleBone_Black,TARGET=pwm1.pru0
- Stopping PRU 0
- copying firmware file /tmp/cloud9-examples/pwm1.pru0.out to
/lib/firmware/am335x-pru0-fw
write_init_pins.sh
- Starting PRU 0
MODEL    = TI_AM335x_BeagleBone_Black
PROC     = pru
PRUN     = 0
PRU_DIR  = /sys/class/remoteproc/remoteproc1
```

Now attach an LED (or oscilloscope) to P9_31 on the Black or P1.36 on the Pocket. You should see a squarewave.

Discussion

Since this is our first example we'll discuss the many parts in detail.

pwm1.pru0.c

[Line-by-line of pwm1.pru0.c](#) is a line-by-line explanation of the c code.

Table 7. Line-by-line of pwm1.pru0.c

Line	Explanation
1	Standard c-header include
2	Include for the PRU. The compiler knows where to find this since the Makefile says to look for includes in /usr/lib/ti/pru-software-support-package
3	The file resource_table_empty.h is used by the PRU loader. Generally we'll use the same file, and don't need to modify it.

Here's what's in `resource_table_empty.h`.`resource_table_empty.c`

```
1  /*
2  * ===== resource_table_empty.h =====
3  *
4  * Define the resource table entries for all PRU cores. This will be
5  * incorporated into corresponding base images, and used by the remoteproc
6  * on the host-side to allocated/reserve resources. Note the remoteproc
7  * driver requires that all PRU firmware be built with a resource table.
8  *
9  * This file contains an empty resource table. It can be used either as:
10 *
11 *     1) A template, or
12 *     2) As-is if a PRU application does not need to configure PRU_INTC
13 *        or interact with the rpmsg driver
14 *
15 */
16
17 #ifndef _RSC_TABLE_PRU_H_
18 #define _RSC_TABLE_PRU_H_
19
20 #include <stddef.h>
21 #include <rsc_types.h>
22
23 struct my_resource_table {
24     struct resource_table base;
25
26     uint32_t offset[1]; /* Should match 'num' in actual definition */
27 };
28
29 #pragma DATA_SECTION(pru_remoteproc_ResourceTable, ".resource_table")
30 #pragma RETAIN(pru_remoteproc_ResourceTable)
31 struct my_resource_table pru_remoteproc_ResourceTable = {
32     1, /* we're the first version that implements this */
33     0, /* number of entries in the table */
34     0, 0, /* reserved, must be zero */
35     0, /* offset[0] */
36 };
37
38 #endif /* _RSC_TABLE_PRU_H_ */
```

More Examples @ <http://beagleboard.org/static/prucookbook/>