PRU Examples

### [Getting Example Code](http://beagleboard.org/static/prucookbook/#_getting_example_code)

### [Problem](http://beagleboard.org/static/prucookbook/#_problem_13)

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

### [Solution](http://beagleboard.org/static/prucookbook/#_solution_13)

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](http://beagleboard.org/static/prucookbook/#_blinking_an_led)

### [Problem](http://beagleboard.org/static/prucookbook/#_problem_14)

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

### [Solution](http://beagleboard.org/static/prucookbook/#_solution_14)

The 'hello, world' of the embedded world is to flash an LED. [hello.pru0.c](http://beagleboard.org/static/prucookbook/#start_hello) is some code that blinks the USR3 LED ten times using the PRU.

*hello.pru0.c*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | #include <stdint.h>  #include <pru\_cfg.h>  #include "resource\_table\_empty.h"  #include "prugpio.h"  volatile register unsigned int *R30;*  *volatile register unsigned int* R31;  void main(void) {  int i;  uint32\_t **gpio1 = (uint32\_t \*)GPIO1;**  **/** Clear SYSCFG[STANDBY\_INIT] to enable OCP master port \*/  CT\_CFG.SYSCFG\_bit.STANDBY\_INIT = 0;  for(i=0; i<10; i++) {  gpio1[GPIO\_SETDATAOUT] = USR3; // The the USR3 LED on  *delay\_cycles(500000000/5); // Wait 1/2 second*  *gpio1[GPIO\_CLEARDATAOUT] = USR3;*  delay\_cycles(500000000/5);  }  \_\_halt();  }  // Turns off triggers  #pragma DATA\_SECTION(init\_pins, ".init\_pins")  #pragma RETAIN(init\_pins)  const char init\_pins[] =  "/sys/class/leds/beaglebone:green:usr3/trigger\0none\0" \  "\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](http://beagleboard.org/static/prucookbook/#_uart)

### [Problem](http://beagleboard.org/static/prucookbook/#_problem_27)

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

### [Solution](http://beagleboard.org/static/prucookbook/#_solution_27)

One simple, yet effective approach to 'printing' from the PRU is an idea taken from the Adruino 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.

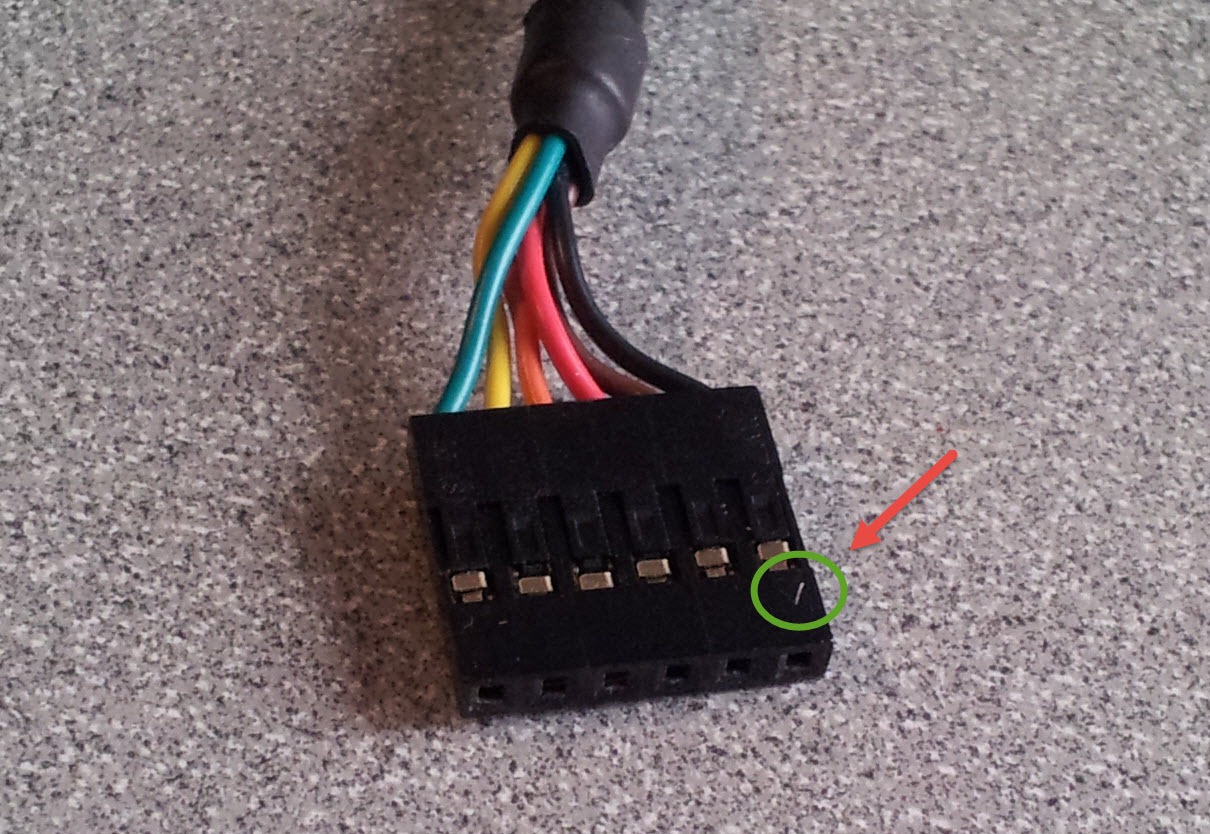
You’ll need a **3.3V FTDI cable** to go between your Beagle and the USB port on your host computer as shown in [FDTI cable](http://beagleboard.org/static/prucookbook/#debug_ftdi).[FTDI images are from the BeagleBone Cookbook <http://shop.oreilly.com/product/0636920033899.do>] You can get such a cable from places such as [Sparkfun](https://www.sparkfun.com/products/9717) or [Adafruit](https://www.adafruit.com/product/70).



*Figure 25. FDTI cable*

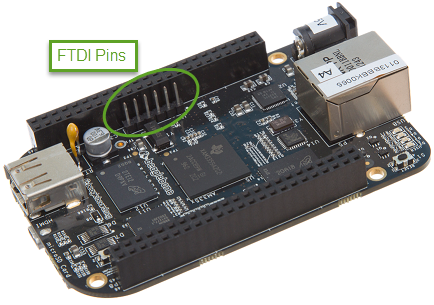
### [Discussion](http://beagleboard.org/static/prucookbook/#_discussion_13)

The Beagle side of the FTDI cable has a small triangle on it as shown in [FTDI connector](http://beagleboard.org/static/prucookbook/#debug_ftdi_connector).



*Figure 26. FTDI connector*

The connector attaches to the Black on the FTDI pins, shown in [FTDI pins for the FTDI connector](http://beagleboard.org/static/prucookbook/#debug_ftdi_pins), 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](http://beagleboard.org/static/prucookbook/#debug_uart1)) 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](http://beagleboard.org/static/prucookbook/#debug_uart2)) 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**](http://beagleboard.org/static/prucookbook/#_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

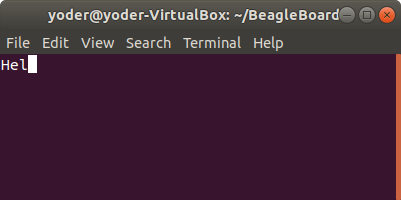
start

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 charters (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  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88 | // From: http://git.ti.com/pru-software-support-package/pru-software-support-package/trees/master/examples/am335x/PRU\_Hardware\_UART  #include <stdint.h>  #include <pru\_uart.h>  #include "resource\_table\_empty.h"  /\* The FIFO size on the PRU UART is 16 bytes; however, we are (arbitrarily)  \* only going to send 8 at a time \*/  #define FIFO\_SIZE 16  #define MAX\_CHARS 8  void main(void)  {  uint8\_t tx;  uint8\_t rx;  uint8\_t cnt;  /\* hostBuffer points to the string to be printed \*/  char\* hostBuffer;  /\*\*\* INITIALIZATION \*\*\*/  /\* Set up UART to function at 115200 baud - DLL divisor is 104 at 16x oversample  \* 192MHz / 104 / 16 = ~115200 \*/  CT\_UART.DLL = 104;  CT\_UART.DLH = 0;  CT\_UART.MDR = 0x0;  /\* Enable Interrupts in UART module. This allows the main thread to poll for  \* Receive Data Available and Transmit Holding Register Empty \*/  CT\_UART.IER = 0x7;  /\* If FIFOs are to be used, select desired trigger level and enable  \* FIFOs by writing to FCR. FIFOEN bit in FCR must be set first before  \* other bits are configured \*/  /\* Enable FIFOs for now at 1-byte, and flush them \*/  CT\_UART.FCR = (0x8) | (0x4) | (0x2) | (0x1);  //CT\_UART.FCR = (0x80) | (0x4) | (0x2) | (0x01); // 8-byte RX FIFO trigger  /\* Choose desired protocol settings by writing to LCR \*/  /\* 8-bit word, 1 stop bit, no parity, no break control and no divisor latch \*/  CT\_UART.LCR = 3;  /\* Enable loopback for test \*/  CT\_UART.MCR = 0x00;  /\* Choose desired response to emulation suspend events by configuring  \* FREE bit and enable UART by setting UTRST and URRST in PWREMU\_MGMT \*/  /\* Allow UART to run free, enable UART TX/RX \*/  CT\_UART.PWREMU\_MGMT = 0x6001;  /\*\*\* END INITIALIZATION \*\*\*/  /\* Priming the 'hostbuffer' with a message \*/  hostBuffer = "Hello! This is a long string\r\n";  /\*\*\* SEND SOME DATA \*\*\*/  /\* Let's send/receive some dummy data \*/  while(1) {  cnt = 0;  while(1) {  /\* Load character, ensure it is not string termination \*/  if ((tx = hostBuffer[cnt]) == '\0')  break;  cnt++;  CT\_UART.THR = tx;  /\* Because we are doing loopback, wait until LSR.DR == 1  \* indicating there is data in the RX FIFO \*/  while ((CT\_UART.LSR & 0x1) == 0x0);  /\* Read the value from RBR \*/  rx = CT\_UART.RBR;  /\* Wait for TX FIFO to be empty \*/  while (!((CT\_UART.FCR & 0x2) == 0x2));  }  }  /\*\*\* DONE SENDING DATA \*\*\*/  /\* Disable UART before halting \*/  CT\_UART.PWREMU\_MGMT = 0x0;  /\* Halt PRU core \*/  \_\_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**](http://beagleboard.org/static/prucookbook/#_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

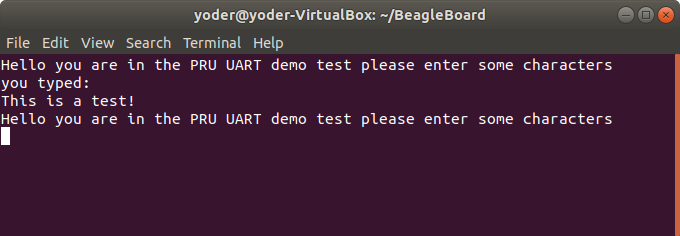
- copying firmware file /tmp/pru0-gen/uart2.out to /lib/firmware/am335x-pru0-fw

- Starting PRU 0

start

You will see:

*uart2.c output*



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](http://beagleboard.org/static/prucookbook/#debug_uart2) is the code that does it.

*uart2.c*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126 | // From: http://git.ti.com/pru-software-support-package/pru-software-support-package/trees/master/pru\_cape/pru\_fw/PRU\_Hardware\_UART  #include <stdint.h>  #include <pru\_uart.h>  #include "resource\_table\_empty.h"  /\* The FIFO size on the PRU UART is 16 bytes; however, we are (arbitrarily)  \* only going to send 8 at a time \*/  #define FIFO\_SIZE 16  #define MAX\_CHARS 8  #define BUFFER 40  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Print Message Out  // This function take in a string literal of any size and then fill the  // TX FIFO when it's empty and waits until there is info in the RX FIFO  // before returning.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  void PrintMessageOut(volatile char\* Message)  {  uint8\_t cnt, index = 0;  while (1) {  cnt = 0;  /\* Wait until the TX FIFO and the TX SR are completely empty \*/  while (!CT\_UART.LSR\_bit.TEMT);  while (Message[index] != NULL && cnt < MAX\_CHARS) {  CT\_UART.THR = Message[index];  index++;  cnt++;  }  if (Message[index] == NULL)  break;  }  /\* Wait until the TX FIFO and the TX SR are completely empty \*/  while (!CT\_UART.LSR\_bit.TEMT);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // IEP Timer Config  // This function waits until there is info in the RX FIFO and then returns  // the first character entered.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  char ReadMessageIn(void)  {  while (!CT\_UART.LSR\_bit.DR);  return CT\_UART.RBR\_bit.DATA;  }  void main(void)  {  uint32\_t i;  volatile uint32\_t not\_done = 1;  char rxBuffer[BUFFER];  rxBuffer[BUFFER-1] = NULL; // null terminate the string  /\*\*\* INITIALIZATION \*\*\*/  /\* Set up UART to function at 115200 baud - DLL divisor is 104 at 16x oversample  \* 192MHz / 104 / 16 = ~115200 \*/  CT\_UART.DLL = 104;  CT\_UART.DLH = 0;  CT\_UART.MDR\_bit.OSM\_SEL = 0x0;  /\* Enable Interrupts in UART module. This allows the main thread to poll for  \* Receive Data Available and Transmit Holding Register Empty \*/  CT\_UART.IER = 0x7;  /\* If FIFOs are to be used, select desired trigger level and enable  \* FIFOs by writing to FCR. FIFOEN bit in FCR must be set first before  \* other bits are configured \*/  /\* Enable FIFOs for now at 1-byte, and flush them \*/  CT\_UART.FCR = (0x80) | (0x8) | (0x4) | (0x2) | (0x01); // 8-byte RX FIFO trigger  /\* Choose desired protocol settings by writing to LCR \*/  /\* 8-bit word, 1 stop bit, no parity, no break control and no divisor latch \*/  CT\_UART.LCR = 3;  /\* If flow control is desired write appropriate values to MCR. \*/  /\* No flow control for now, but enable loopback for test \*/  CT\_UART.MCR = 0x00;  /\* Choose desired response to emulation suspend events by configuring  \* FREE bit and enable UART by setting UTRST and URRST in PWREMU\_MGMT \*/  /\* Allow UART to run free, enable UART TX/RX \*/  CT\_UART.PWREMU\_MGMT\_bit.FREE = 0x1;  CT\_UART.PWREMU\_MGMT\_bit.URRST = 0x1;  CT\_UART.PWREMU\_MGMT\_bit.UTRST = 0x1;  /\* Turn off RTS and CTS functionality \*/  CT\_UART.MCR\_bit.AFE = 0x0;  CT\_UART.MCR\_bit.RTS = 0x0;  /\*\*\* END INITIALIZATION \*\*\*/  while(1) {  /\* Print out greeting message \*/  PrintMessageOut("Hello you are in the PRU UART demo test please enter some characters\r\n");  /\* Read in characters from user, then echo them back out \*/  for (i = 0; i < BUFFER-1 ; i++) {  rxBuffer[i] = ReadMessageIn();  if(rxBuffer[i] == '\r') { // Quit early if ENTER is hit.  rxBuffer[i+1] = NULL;  break;  }  }  PrintMessageOut("you typed:\r\n");  PrintMessageOut(rxBuffer);  PrintMessageOut("\r\n");  }  /\*\*\* DONE SENDING DATA \*\*\*/  /\* Disable UART before halting \*/  CT\_UART.PWREMU\_MGMT = 0x0;  /\* Halt PRU core \*/  \_\_halt();  } |

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

### [PWM Generator](http://beagleboard.org/static/prucookbook/" \l "blocks_pwm)

One of the simplest things a PRU can to is generate a simple signals 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](http://beagleboard.org/static/prucookbook/#_problem_30)

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

### [Solution](http://beagleboard.org/static/prucookbook/#_solution_30)

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

[pwm1.pru0.c](http://beagleboard.org/static/prucookbook/#blocks_pwm1) shows the code.

*pwm1.pru0.c*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | #include <stdint.h>  #include <pru\_cfg.h>  #include "resource\_table\_empty.h"  #include "prugpio.h"  volatile register uint32\_t \_\_R30;  volatile register uint32\_t \_\_R31;  void main(void)  {  uint32\_t gpio = P9\_31; // Select which pin to toggle.;  /\* Clear SYSCFG[STANDBY\_INIT] to enable OCP master port \*/  CT\_CFG.SYSCFG\_bit.STANDBY\_INIT = 0;  while(1) {  \_\_R30 |= gpio; // Set the GPIO pin to 1  \_\_delay\_cycles(100000000);  \_\_R30 &= ~gpio; // Clear the GPIO pin  \_\_delay\_cycles(100000000);  }  } |

* 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 pruout**

On the Pocket run

bone$ **config-pin P1\_36 pruout**

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](http://beagleboard.org/static/prucookbook/#_discussion_16)

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

##### [**pwm1.pru0.c**](http://beagleboard.org/static/prucookbook/#_pwm1_pru0_c)

[Line-by-line of pwm1.pru0.c](http://beagleboard.org/static/prucookbook/#blocks_pwm1_line_by_line) is a line-by-line expanation 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  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | /\*  \* ======== resource\_table\_empty.h ========  \*  \* Define the resource table entries for all PRU cores. This will be  \* incorporated into corresponding base images, and used by the remoteproc  \* on the host-side to allocated/reserve resources. Note the remoteproc  \* driver requires that all PRU firmware be built with a resource table.  \*  \* This file contains an empty resource table. It can be used either as:  \*  \* 1) A template, or  \* 2) As-is if a PRU application does not need to configure PRU\_INTC  \* or interact with the rpmsg driver  \*  \*/  #ifndef \_RSC\_TABLE\_PRU\_H\_  #define \_RSC\_TABLE\_PRU\_H\_  #include <stddef.h>  #include <rsc\_types.h>  struct my\_resource\_table {  struct resource\_table base;  uint32\_t offset[1]; /\* Should match 'num' in actual definition \*/  };  #pragma DATA\_SECTION(pru\_remoteproc\_ResourceTable, ".resource\_table")  #pragma RETAIN(pru\_remoteproc\_ResourceTable)  struct my\_resource\_table pru\_remoteproc\_ResourceTable = {  1, /\* we're the first version that implements this \*/  0, /\* number of entries in the table \*/  0, 0, /\* reserved, must be zero \*/  0, /\* offset[0] \*/  };  #endif /\* \_RSC\_TABLE\_PRU\_H\_ \*/ |

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