Ti AM33XX PRUSSv2

From eLinux.org

The PRUSS (Programmable Real-time Unit Sub System) consists of two 32-bit 200MHz real-time cores, each with 8KB of program memory and direct access to general I/O. These cores are connected to various data memories, peripheral modules and an interrupt controller for access to the entire system-on-a-chip via a 32-bit interconnect bus.

PRUs are programmed in Assembly (http://en.wikipedia.org/wiki/Assembly_language), with most commands executing in a single cycle and with no caching or pipe-lining, allowing for 100% predictable timings. At 200MHz, most operations will take 5ns (nanoseconds) to execute, with the exception of accessing memory external to PRU.

Contents

- 1 This is a Work In Progress
- 2 Available PRU Resources
  - 2.1 Per PRU
  - 2.2 Shared Between PRUs
  - 2.3 Local Peripherals
- 3 Communication
  - 3.1 PRU to Host (PRU to ARM Cortex-A8)
  - 3.2 Host to PRU (ARM Cortex-A8 to PRU)
    - 3.2.1 Interrupts
  - 3.3 PRU to external peripherals
  - 3.4 External peripherals to PRU
  - 3.5 PRU to internal peripherals
  - 3.6 Internal peripherals to PRU
- 4 Loading a PRU Program
- 5 Beaglebone PRU connections and modes
- 6 Assembly
  - 6.1 Four instruction classes
  - 6.2 Instruction Syntax
- 7 C Compiler
  - 7.1 TI
  - 7.2 GCC
- 8 Forth Compiler
- 9 Resources
- 10 Examples

This is a Work In Progress

Available PRU Resources
Click here for a full list of register mappings.

**Per PRU**

**8KB program memory**

Memory used to store instructions and static data AKA Instruction Memory (IRAM). This is the memory in which PRU programs are loaded.

**Enhanced GPIO (EGPIO)**

High-speed direct access to 16 general purpose output and 17 general purpose input pins for each PRU.

- **PRU0**
  - **pr1_pru_0_pru_r30[15:0]** (PRU0 Register R30 Outputs)
  - **pr1_pru_0_pru_r31[16:0]** (PRU0 Register R31 Inputs)

- **PRU1**
  - **pr1_pru_1_pru_r30[15:0]** (PRU1 Register R30 Outputs)
  - **pr1_pru_1_pru_r31[16:0]** (PRU1 Register R31 Inputs)

**Hardware capture modes**

- Serial 28-bit shift in and out.
- Parallel 16-bit capture on clock.
- MII (http://en.wikipedia.org/wiki/Media_Independent_Interface) standardised capture mode, used for implementing media independent Fast Ethernet (100Mbps - 25MHz 4-bit).

**A 32-bit multiply and accumulate unit (MAC)**

Enables single-cycle integer multiplications with a 64-bit overflow (useful for decimal results).

**8KB data memory**

Memory used to store dynamic data. Is accessed over the 32-bit bus and so not single-cycle.

One PRU may access the memory of another for passing information but it is recommend to use scratch pad or shared memory, see below.

**Open Core Protocol (http://en.wikipedia.org/wiki/Open_Core_Protocol) (OCP) master port**

Access to the data bus that interconnects all peripherals on the SoC, including the ARM Cortex-A8, used for data
Shared Between PRUs

**Scratch pad**
3 banks of 30 32-bit registers (total 90 32-bit registers).
Single-cycle access, can be accessed from either PRU for data sharing and signalling or for individual use.

**12KB data memory**
Accessed over the 32-bit bus, not single-cycle.

Local Peripherals

Local peripherals are those present within the PRUSS and not those belonging to the entire SoC. Peripherals are accessed from PRUs over the Switched Central Resource (SCR) 32-bit bus within the PRUSS.

Attached to the SCR bus is also an OCP slave, enabling OCP masters from outside of the PRUSS to access these local peripherals in Level 4 (L4) memory space.

**Enhanced Capture Model (eCAP)**

**Industrial Ethernet Peripheral (IEP)**

**Universal Asynchronous Receiver/Transmitter (UART0)**
Used to perform serial data transmission to the TL16C550 industry standard.
16-bit FIFO receive and transmit buffers + per byte error status.
Can generate Interrupt requests for the PRUSS Interrupt Controller.
Can generate DMA requests for the EDMA SoC DMA controller.
Maximum transmission speed of 192MHz (192Mbps - 24MB/s).

Communication

Communication between various elements of the PRUSS or the wider SoC may take place either directly, over a bus, via interrupts or via DMA.

The following lists will expose all possible communication approaches for each likely scenario.

For communication via interrupts, please first read the section on the PRUSSv2 Interrupt Controller.

Click here for a full list of PRUSS Interrupts.

The current example PRU loader (https://github.com/beagleboard/am335x_pru_package/blob/master/pru_sw/app_loader/interface/prussdrv.c) uses UIO (http://arago-project.org/git/projects/?p=linux-am33x.git;a=commit;h=f1a304e7941cc76353363a139cbb6a4b1ca7c737), but this ideally should be replaced with remoteproc (http://omappedia.org/wiki/Category:RPMsg) rather than poking at the registers from userspace. In the mean time, according to this discussion: (https://groups.google.com/d/msg/beagleboard/gqCjxh4uZi0/_uIUD8ZF88QJ) we can use the included script and load the uio_pruss userspace driver.

**PRU to Host (PRU to ARM Cortex-A8)**

Include the uio_pruss kernel driver by using `modprobe uio_pruss` or the steps outlined above, if that does not work. Then in a project include the header files for the am335x_pru_package.

```c
#define PRU_NUM0 0
// Driver header file
#include <prussdrv.h>
#include <pruss_intc_mapping.h>
```

http://elinux.org/Ti_AM33XX_PRUSSv2
/* Then, initialize the interrupt controller data */

tpruss_intc_initdata pruss_intc_initdata = PRUSS_INTC_INITDATA;

/* Initialize the PRU */

prussdrv_init();

/* Get the interrupt initialized */

prussdrv_pruintc_init(&pruss_intc_initdata)

/* Execute example on PRU0 where first argument is the PRU# and second is the assembly to execute*/

prussdrv_exec_program (PRU_NUM0, "./example.bin");

/* Wait until PRU0 sends the interrupt*/

prussdrv_pru_wait_event (PRU_EVTOUT_0);

/* Clear the interrupt*/

prussdrv_pru_clear_event (PRU0_ARM_INTERRUPT);

The PRU (in this case 0) will have the following in the example.bin file to trigger the interrupt:

#define PRU0_ARM_INTERRUPT 19
MOV r31.b0, PRU0_ARM_INTERRUPT+16

Register 31 allows for control of the INTC for the PRU.

**Host to PRU (ARM Cortex-A8 to PRU)**

**Interrupts**

Each PRU has access to host interrupt channels Host-0 and Host-1 through register R31 bit 30 and bit 31 respectively. By probing these registers, a PRU can determine if an interrupt is currently present on each host channel.

To configure

**PRU to external peripherals**

**External peripherals to PRU**

**PRU to internal peripherals**

**Internal peripherals to PRU**
## Loading a PRU Program

### Beaglebone PRU connections and modes

<table>
<thead>
<tr>
<th>PRU #</th>
<th>R30(output) bit</th>
<th>R31(input) bit</th>
<th>Pinmux Mode</th>
<th>PINMUX Mode</th>
<th>BB Header</th>
<th>BB Pin Name</th>
<th>ZCZ BallName</th>
<th>Offset Reg</th>
<th>DT Offset</th>
<th>Input Mode</th>
<th>Output Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Mode_5</td>
<td>SPI1_SCLK</td>
<td>P9_31</td>
<td>mcsap0_aclkx</td>
<td>990h</td>
<td>0x190</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Mode_5</td>
<td>SPI1_D0</td>
<td>P9_29</td>
<td>mcsap0_fsx</td>
<td>994h</td>
<td>0x194</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>Mode_5</td>
<td>SPI1_D1</td>
<td>P9_30</td>
<td>mcsap0_axr0</td>
<td>998h</td>
<td>0x198</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>3</td>
<td>Mode_5</td>
<td>SPI1_CS0</td>
<td>P9_28</td>
<td>mcsap0_aclk</td>
<td>99Ch</td>
<td>0x19C</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>4</td>
<td>Mode_5</td>
<td>SPI1_CS0</td>
<td>P9_42</td>
<td>mcsap0_aclk</td>
<td>9A0h</td>
<td>0x1A0</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>5</td>
<td>Mode_5</td>
<td>GPIO3_19</td>
<td>P9_27</td>
<td>mcsap0_fs</td>
<td>9A4h</td>
<td>0x1A4</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>6</td>
<td>Mode_5</td>
<td>GPIO3_21</td>
<td>P9_41</td>
<td>mcsap0_axr1</td>
<td>9A8h</td>
<td>0x1A8</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>7</td>
<td>Mode_5</td>
<td>GPIO3_23</td>
<td>P9_25</td>
<td>mcsap0_aclk</td>
<td>9ACh</td>
<td>0x1AC</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>14</td>
<td>N/A</td>
<td>Mode_6</td>
<td>GPIO1_12</td>
<td>P8_12</td>
<td>gpme_ad12</td>
<td>830h</td>
<td>0x030</td>
<td>N/A</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>15</td>
<td>N/A</td>
<td>Mode_6</td>
<td>GPIO1_13</td>
<td>P8_11</td>
<td>gpme_ad13</td>
<td>834h</td>
<td>0x034</td>
<td>N/A</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>N/A</td>
<td>14</td>
<td>Mode_6</td>
<td>GPIO1_14</td>
<td>P8_16</td>
<td>gpme_ad14</td>
<td>838h</td>
<td>0x038</td>
<td>N/A</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>N/A</td>
<td>15</td>
<td>Mode_6</td>
<td>GPIO1_15</td>
<td>P8_15</td>
<td>gpme_ad15</td>
<td>83Ch</td>
<td>0x03C</td>
<td>N/A</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>N/A</td>
<td>16</td>
<td>Mode_6</td>
<td>UART1_TXD</td>
<td>P9_24</td>
<td>uart1_rxd</td>
<td>984h</td>
<td>0x184</td>
<td>N/A</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Mode_5</td>
<td>GPIO2_6</td>
<td>P8_45</td>
<td>lcd_data0</td>
<td>8A0h</td>
<td>0x0A0</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Mode_5</td>
<td>GPIO2_7</td>
<td>P8_46</td>
<td>lcd_data1</td>
<td>8A4h</td>
<td>0x0A4</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>Mode_5</td>
<td>GPIO2_8</td>
<td>P8_43</td>
<td>lcd_data2</td>
<td>8A8h</td>
<td>0x0A8</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>Mode_5</td>
<td>GPIO2_9</td>
<td>P8_44</td>
<td>lcd_data3</td>
<td>8ACh</td>
<td>0x0AC</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>Mode_5</td>
<td>GPIO2_10</td>
<td>P8_41</td>
<td>lcd_data4</td>
<td>8B0h</td>
<td>0x0B0</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
<td>Mode_5</td>
<td>GPIO2_11</td>
<td>P8_42</td>
<td>lcd_data5</td>
<td>8B4h</td>
<td>0x0B4</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>6</td>
<td>Mode_5</td>
<td>GPIO2_12</td>
<td>P8_39</td>
<td>lcd_data6</td>
<td>8B8h</td>
<td>0x0B8</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>7</td>
<td>Mode_5</td>
<td>GPIO2_13</td>
<td>P8_40</td>
<td>led_data7</td>
<td>8BCh</td>
<td>0x0BC</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>8</td>
<td>Mode_6</td>
<td>GPIO2_14</td>
<td>P8_27</td>
<td>led_vsync</td>
<td>8E0h</td>
<td>0x0E0</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>9</td>
<td>Mode_6</td>
<td>GPIO2_15</td>
<td>P8_29</td>
<td>led_hsync</td>
<td>8E4h</td>
<td>0x0E4</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>Mode_6</td>
<td>GPIO2_16</td>
<td>P8_28</td>
<td>led_pelk</td>
<td>8E8h</td>
<td>0x0E8</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>11</td>
<td>Mode_6</td>
<td>GPIO2_17</td>
<td>P8_30</td>
<td>led_ac_bias_en</td>
<td>8EC</td>
<td>0x0EC</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>12</td>
<td>Mode_6</td>
<td>GPIO1_18</td>
<td>P8_21</td>
<td>gpme_cs1</td>
<td>880h</td>
<td>0x080</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>13</td>
<td>Mode_6</td>
<td>GPIO1_19</td>
<td>P8_20</td>
<td>gpme_cs2</td>
<td>884h</td>
<td>0x084</td>
<td>0x06</td>
<td>0x25</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>N/A</td>
<td>16</td>
<td>Mode_6</td>
<td>UART1_RXD</td>
<td>P9_26</td>
<td>uart1_rxd</td>
<td>980h</td>
<td>0x180</td>
<td>0x06</td>
<td>NA</td>
<td></td>
</tr>
</tbody>
</table>

*Note1: The PRU0 Registers(30,31) Bit 4 (GPIO3_18) is routed to P9_42-GPIOB_7 pin. You **MUST** set GPIO0_7 to input mode in pinmuxing.

*Note2: The PRU0 Registers(30,31) Bit 6 (GPIO3_20) is routed to P9_41-GPIOB_20(CLKOUT2). You **MUST** set GPIO0_20 to input mode in pinmuxing.

---

## Assembly

The complete list of PRU assembly instructions can be found at TI (http://processors.wiki.ti.com/index.php/PRU_Assembly_Instructions).
Four instruction classes

- Arithmetic
- Logical
- Flow Control
- Register Load/Store

Instruction Syntax

- Mnemonic, followed by comma separated parameter list
- Parameters can be a register, label, immediate value, or constant table entry
- Example
  - SUB r3, r4, 10
  - Subtracts immediate value 10 (decimal) from the value in r4 and then places the result in r3 (or r3 = r4 - 10)

C Compiler

TI

- TI C Compiler download (http://software-dl.ti.com/codegen/non-esd/downloads/download.htm#PRU)
- TI Code Composer Studio (http://www.ti.com/tool/ccstudio-sitara)
- CCS PRU compiler release discussion on Element14 site

GCC

- GCC C wiki (https://github.com/dinuxbg/gnupru/wiki)
- GCC C compiler source (https://github.com/dinuxbg/gnupru)
- GCC C compiler announcement (https://groups.google.com/forum/#!topic/beagleboard/hC5OwPXuSmQ)

Forth Compiler

- Forth for PRU (https://github.com/biocode3D/prufh)

Resources

- PRU FAQ (https://groups.google.com/forum/?fromgroups#!topic/beagleboard/u28ytaoNenU)
- AM335x PRU support package on Github (https://github.com/beagleboard/am335x_pru_package)
- AM335x PRU-ICSS Reference Guide
  (https://github.com/beagleboard/am335x_pru_package/blob/master/am335xPruReferenceGuide.pdf?raw=true)
- The documentation on the subsystem is here. TI does not support this subsystem and all questions/inquires/problems should be directed to the community.
- Example for the first PRU version (Ti AM18XX and other DSP) http://www.ti.com/tool/sprc940

- PRU assembly syntax highlighting for TextMate, Sublime Text, etc. (https://github.com/sagedevices/ti-pruv2-assembly-textmate-bundle)

- A userspace debugging utility (https://docs.google.com/file/d/0B80aJokrBccAV1paMTU0bjM1OXc/edit?usp=sharing) with credit to PRU_EVTOUT_2 from the #beagle IRC channel.
- ncurses based debugger work has started at here (https://github.com/wz2b/prude).
- A classic CLI debugger is available on SourceForge called prudebug (http://sourceforge.net/projects/prudebug).
- For using the Open Core Protocol to access external memory (http://nomel.org/post/30006622413/beaglebone-tutorial-accessing-main-memory-from-the-pru) from the PRU.
Examples

- BeagleBone_6502_RemoteProc_cape
- BeagleBone Nixie Cape PRU App (https://github.com/mranostay/beagle-nixie)
- SPI ADC (http://exploringbeaglebone.com/chapter13/#prettyPhoto)


Categories: ECE497 | BeagleBone | PRU

- This page was last modified on 12 March 2015, at 07:05.
- Content is available under a Creative Commons Attribution-ShareAlike 3.0 Unported License unless otherwise noted.