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.

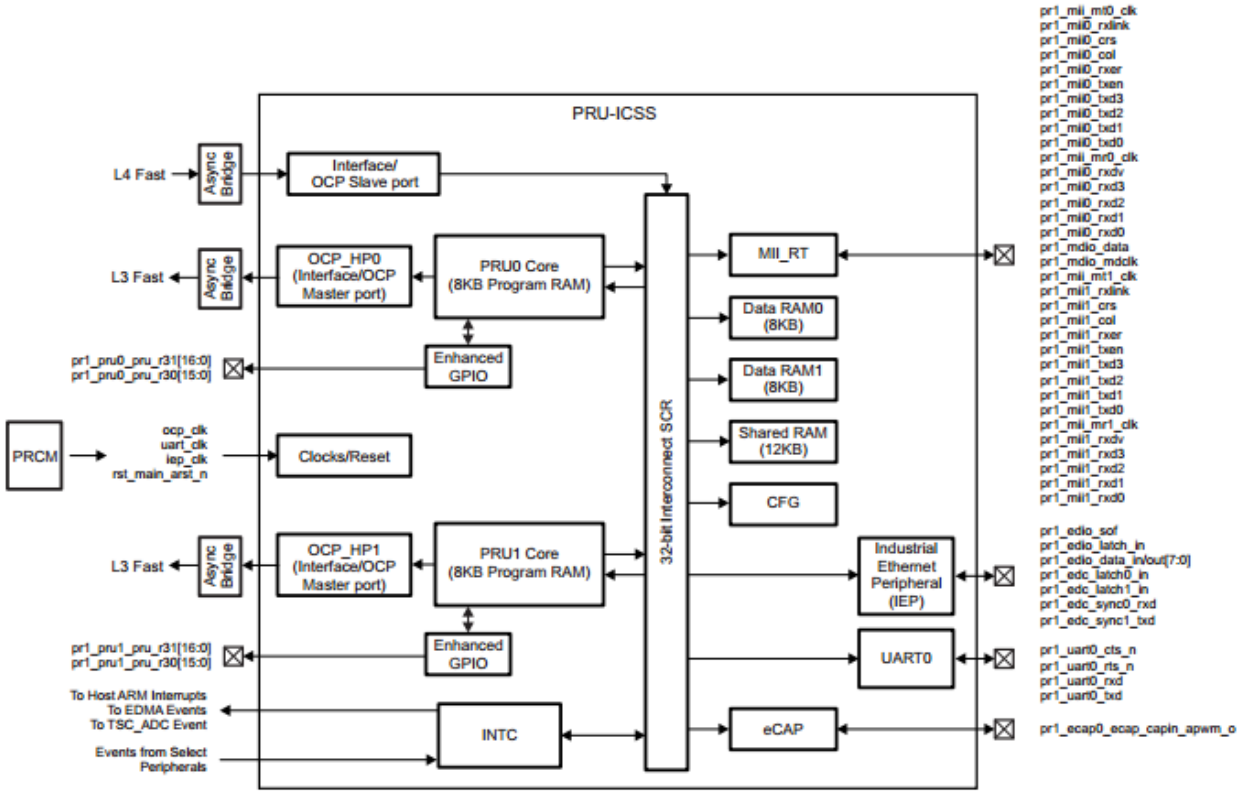
PRUs are programmed in Assembly (http://en.wikipedia.org/wiki/Assembly_language), with most commands executing in a single cycle and with no caching or pipe-lining, allowing for 100% predictable timings. At 200MHz, most operations will take 5ns (nanoseconds) to execute, with the exception of accessing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

Contents

This is a Work In Progress

Available PRU Resources

Figure 2. PRU-ICSS Integration



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

transfer directly to and from the PRU in Level 3 (L3) memory space.

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-but 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.

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

/* 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

PRU #	R30(output) bit	Pinmux Mode	R31(input) bit	Pinmux Mode	BB Header	BB Pin Name	ZCZ BallName	Offset Reg	DT Offset	Input Mode	Output Mode
0	0	Mode_5	0	Mode_6	P9_31	SPI1_SCLK	mcasp0_aclkx	990h	0x190	0x06	0x25
0	1	Mode_5	1	Mode_6	P9_29	SPI1_D0	mcasp0_fsx	994h	0x194	0x06	0x25
0	2	Mode_5	2	Mode_6	P9_30	SPI1_D1	mcasp0_axr0	998h	0x198	0x06	0x25
0	3	Mode_5	3	Mode_6	P9_28	SPI1_CS0	mcasp0_ahclkr	99Ch	0x19C	0x06	0x25
0	4	Mode_5	4	Mode_6	P9_42	(*note1)	mcasp0_aclkr	9A0h	0x1A0	0x06	0x25
0	5	Mode_5	5	Mode_6	P9_27	GPIO3_19	mcasp0_fsr	9A4h	0x1A4	0x06	0x25
0	6	Mode_5	6	Mode_6	P9_41	(*note2)	mcasp0_axr1	9A8h	0x1A8	0x06	0x25
0	7	Mode_5	7	Mode_6	P9_25	GPIO3_21	mcasp0_ahclkx	9ACh	0x1AC	0x06	0x25
0	14	Mode_6	N/A		P8_12	GPIO1_12	gpmc_ad12	830h	0x030	N/A	0x25
0	15	Mode_6	N/A		P8_11	GPIO1_13	gpmc_ad13	834h	0x034	N/A	0x25
0	N/A		14	Mode_6	P8_16	GPIO1_14	gpmc_ad14	838h	0x038	0x06	N/A
0	N/A		15	Mode_6	P8_15	GPIO1_15	gpmc_ad15	83Ch	0x03C	0x06	N/A
0	N/A		16	Mode_6	P9_24	UART1_TXD	uart1_txd	984h	0x184	0x06	N/A
1	0	Mode_5	0	Mode_6	P8_45	GPIO2_6	lcd_data0	8A0h	0x0A0	0x06	0x25
1	1	Mode_5	1	Mode_6	P8_46	GPIO2_7	lcd_data1	8A4h	0x0A4	0x06	0x25
1	2	Mode_5	2	Mode_6	P8_43	GPIO2_8	lcd_data2	8A8h	0x0A8	0x06	0x25
1	3	Mode_5	3	Mode_6	P8_44	GPIO2_9	lcd_data3	8ACh	0x0AC	0x06	0x25
1	4	Mode_5	4	Mode_6	P8_41	GPIO2_10	lcd_data4	8B0h	0x0B0	0x06	0x25
1	5	Mode_5	5	Mode_6	P8_42	GPIO2_11	lcd_data5	8B4h	0x0B4	0x06	0x25
1	6	Mode_5	6	Mode_6	P8_39	GPIO2_12	lcd_data6	8B8h	0x0B8	0x06	0x25
1	7	Mode_5	7	Mode_6	P8_40	GPIO2_13	lcd_data7	8BCh	0x0BC	0x06	0x25
1	8	Mode_5	8	Mode_6	P8_27	GPIO2_22	lcd_vsync	8E0h	0x0EO	0x06	0x25
1	9	Mode_5	9	Mode_6	P8_29	GPIO2_23	lcd_hsync	8E4h	0x0E4	0x06	0x25
1	10	Mode_5	10	Mode_6	P8_28	GPIO2_24	lcd_pclk	8E8h	0x0E8	0x06	0x25
1	11	Mode_5	11	Mode_6	P8_30	GPIO2_25	lcd_ac_bias_en	8ECh	0x0EC	0x06	0x25
1	12	Mode_5	12	Mode_6	P8_21	GPIO1_30	gpmc_csn1	880h	0x080	0x06	0x25
1	13	Mode_5	13	Mode_6	P8_20	GPIO1_31	gpmc_csn2	884h	0x084	0x06	0x25
1	N/A		16	Mode_6	P9_26	UART1_RXD	uart1_rxd	980h	0x180	0x06	NA

```
*Note1: The PRU0 Registers{30,31} Bit 4 (GPIO3_18) is routed to P9_42-GPIO0_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-GPIO0_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 (http://www.element14.com/community/community/knode/single-board_computers/next-gen_beaglebone/blog/2014/04/30/bbb--pru-c-compiler)

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
- Element14 write-up on using PRU (http://www.element14.com/community/community/knode/single-board_computers/next-gen_beaglebone/blog/2013/05/22/bbb--working-with-the-pru-icssprussv2)

- 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.

- Jason Kridner's presentation at Pumping Station: One - video (http://videos.pumpingstationone.org/video/27/real-time-programming-with-beaglebone-prus) slides (http://beagleboard.org/static/PumpingStationOne20140628_Real-timeProgrammingWithBeagleBonePRUs.pptx.pdf)

Examples

- BeagleBone_6502_RemoteProc_cape
- BeagleBone Nixie Cape PRU App (https://github.com/mranostay/beagle-nixie)
- PRU Soft UART Driver (http://processors.wiki.ti.com/index.php/Soft-UART_Implementation_on_AM335X_PRU_-_Software_Users_Guide)
- PRU Projects (http://processors.wiki.ti.com/index.php/PRU_Projects)
- SPI ADC (http://exploringbeaglebone.com/chapter13/#prettyPhoto)

Categories: ECE497 | BeagleBone | PRU
