Understanding BBB PRU shared memory access

Posted on August 27, 2013 by Owen — 3 Comments ↓

I have previously posted about using the Programmable Realtime Units (PRUs) on the BBB for servo control, and in the PRU code for this I noted that I'd stolen the set up code from the examples provided here. This stolen code enables the PRU OCP ports (enabling communication between the PRUs and the host processor) and sets up the shared memory access. Working with PRU0 this code worked fine, but when I attempted to write code for PRU1 (for an extension to my Wireless Servo Control project), the memory access was not working. After initially suspecting my C++ interface code, I eventually traced this to poorly explained register access by the examples. So I'm writing this post to clear things up for future reference, and to help anyone else who might be stuck!

Example Code Explanation

First, lets have a look at what the example is actually doing. You can refer to my servo control post for the full code, but here is the relevant part.

```
LBCO    r0, CONST_PRUCFG, 4, 4      // Enable OCP master port
CLR     r0, r0, 4                   // Clear SYSCFG[STANDBY_INIT] to ena
SBCO    r0, CONST_PRUCFG, 4, 4
MOV     r0, 0x00000120              // Configure the programmable pointe
MOV     r1, CTPPR_0                 // field to 0x0120.  This will make 
ST32    r0, r1
```

The comments give you an idea of what is going on, but the details are hidden from view. Two important things to note immediately are the constants *CONST_PRUCFG* and *CTPPR_0* and the macro *ST32*. These are defined in the header file provided with the examples (this file has various names depending on your source, I call it pru.hp). To both simplify and complicate things, lets replace these with their values (and strip out the comments/shorten the hex)

```
1  LBCO    r0, C4, 4, 4
2  CLR     r0, r0, 4
3  SBCO    r0, C4, 4, 4
4  MOV     r0, 0x120
5  MOV     r1, 0x22028
6  SBBO    r0, r1, 0, 4
```

Now this immediately looks more complicated, but at least it is easier to relate to the Reference Guide. Let's look at it in some detail

- Line 1 uses LBCO to load bytes 4-8 from the register defined by constant C4 in to local register r0. Table 9 on page 25 of the reference guide tells us that the register defined in C4 is PRU-ICSS CFG, the configuration registers for the PRU system. The details of these registers can be found in Section 10 (starting on page 272). Section 10.1.2 tells us that bytes 4-8 are the SYSCFG register.
- Line 2 clears bit 4 of local register r0. Referring to Section 10.1.2, this is the STANDBY_INIT bit. Clearing this bit disables standby and enables the OCP ports.
- Line 3 simply writes r0 back to the SYSCFG register, enabling the OCP ports.

At this point you might be thinking this is exactly what the comments said, in a much more concise fashion, and you'd be right! However, this is only because the PRU-ICSS CFG register is common to both PRUs. It's the next part that gets tricky

- Line 4 simply moves the hex value 0x120 in to local register r0. This will correspond to the value of the c28_pointer bits of the CTPPR0 register.
- Line 5 moves 0x22028 in to local register r1. This is the address of the CTPPR0 register ***for PRU0***, and this is where all the problems come from. The constant defined previously was named *CTPPR_0*, and there is another one called *CTPPR_1*. Surely these are for PRU0 and PRU1, right? No such luck, both PRU0 and PRU1 have both of these registers, and the constants defined in the examples ***only refer to PRU0***.
- Line 6 simply writes the hex value to the register.

Okay, so we know there is a problem with the register address. But also, where on earth does the value 0x120 come from? Let's figure both of these things out...

CTPPR0 Register Address

The reference guide defines the CTPPR0 register in Section 5.4.8 (page 84) as being at "Offset 28h". Now this is interesting when looking at the hex address above, as it ends in 0x28. Now referring to Table 5 on page 19, we see that the PRU0 control registers start at 0x22000. So there we have it; 0x22028 is the CTPPR0 register for PRU0. Table 5 also informs us that the control registers for PRU1 start on 0x24000, so the CTPPR0 register for 0x24028.

Now we have the address sorted, what about the value.

CTPPR0 Register Value

The reference guide defines the CTPPR0 register as being the Constant Table Programmable Pointer Register 0. It is responsible for setting up the offset of C28 (and C29, which we don't care about) in the constant table. Doing this enables us to use LBCO and SBCO to access memory directly without having to worry about any offsets inside the code.

Referring to the CTPPR0 definition and Table 9 (page 25) we can see that

```
c28_pointer = CTPPR0[0:15]
nnnn = c28_pointer
C28 = 0x00nnnn00
```

Therefore, the value 0x120 used above would set C28 to 0x00012000, so what does this mean?

The global memory map is shown in Table 6 (page 20) and identifies 0x00010000 as being the start of the 12Kb of shared memory. Therefore, value of 0x00012000 is somewhere inside the shared memory (albeit not at the beginning). This explains the somewhat arbitrary 2048 offset inside the C code of the examples, as 0x2000 = 4 * 2048, and the C code reads memory as integers (which are 4 bytes wide).

Now we understand what the value means, it would seem to make more sense to set CTPPR0 to 0x100, corresponding to the start of the shared memory. This would also do away with the need for an offset of 2048 in the host code.

Delving a bit deeper in to the memory map lets us find a few more possible values. The default value of 0x000 corresponds to the PRUs own memory space. This means if you don't change anything, C28 will point at each PRUs own memory. Two PRUs running the exact same code would each be using their own memory spaces and wouldn't cause any conflicts. A value of 0x020 corresponds to the memory space of the other PRU, so this could be used as a form of inter-PRU communication (or just use the shared space).

Some new set up code

Now we've got our head around everything, lets define some more convenient set up code

```
// Put this in a header file
#define CONST_PRUCFG         C4
#define CONST_PRUSHAREDRAM   C28

#define PRU0_CTRL            0x22000
#define PRU1_CTRL            0x24000

```

```
 8  #define CTPPR0              0x28
 9
10  #define OWN_RAM             0x000
11  #define OTHER_RAM           0x020
12  #define SHARED_RAM          0x100
13
14  // Start your code with this
15  LBCO    r0, CONST_PRUCFG, 4, 4          // Enable OCP master port
16  CLR     r0, r0, 4
17  SBCO    r0, CONST_PRUCFG, 4, 4
18  MOV     r0, SHARED_RAM                  // Set C28 to point to shared R
19  MOV     r1, PRU1_CTRL + CTPPR0
20  SBBO    r0, r1, 0, 4
21
22  // Do something useful with memory
```

I have defined some new constants to make everything much easier to follow, namely *PRU0_CTRL* and *PRU1_CTRL* which refer to the base of each PRUs control register. I have then modified the CTPPR0 constant to refer to the offset, not an absolute value. In addition, I have included the three most useful values for the c28_pointer bits of the CTPPR0 register, *OWN_RAM*, *OTHER_RAM* and *SHARED_RAM*.

This set up example is similar to the previous one except it uses the start of the shared memory without the offset of 2048 integers. If you wanted to maintain this offset (as I am currently doing for legacy reasons, until I update my C++ class) you could simply add it to line 18

```
18  MOV     r0, SHARED_RAM + 0x020
```

‹ Wireless Servo Control - Part 3: PWM servo control with the BBB PRU

Wireless Servo Control - Part 3.1: Failsafe servo control ›

Posted in BeagleBone Black

3 comments on “Understanding BBB PRU shared memory access”

Simon says:
June 25, 2014 at 13:51

I had the exact problem, and this solved it. I had looked at the offsets and programmable pointers several times, and could not find anything wrong. I didn't realize there was a register for both PRU's before coming across your post.

Thank you so much for sharing this!

Reply

longqi says:
October 3, 2014 at 09:27

Iam confused about this aprt
c28_pointer = CTPPR0[0:15]
nnnn = c28_pointer
C28 = 0x00nnnn00
why the C28 is not 0x0000nnnn? is it predefined in hardware?

Reply

longqi says:
January 9, 2015 at 08:50

Thanks you so much,
Your article really help me a lot about the shared memory.

Thanks again.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Name *
longqi

E-mail *
benhuan@126.com

Website
http://longqi.pro

[ ] + two = 3

Comment

Post Comment