TMS320 DSP
DESIGNER'S NOTEBOOK
Number 51

Bootloading a 'C4x Network—Part 1: Direct Connect System

Contributed by Gerald Capwell

Design Problem

If the 'C4x devices are directly connected to each other, how can you perform an automatic system boot-up at hardware reset?

Solution

In this case the system is called a "Known, Direct Connect" network. In order to configure a network in this format, the network must have the following criteria:

- The system has one dedicated device known as the system "Parent." The parent device is configured in hardware via the IIOFx pins to boot from external memory (see section 13.2 of the TMS320C4x User's Guide) or from a PC platform. The RESETLOC(1,0) pins must be low in order for the on-chip ROM bootloader to load programs from external memory.
- All devices (except the system parent) are called system "Children." The system child is dependent upon the parent for bootloading. The system children are configured in hardware via the IIOFx pins to boot from commports (see section 13.2 of the TMS320C4x User's Guide).
- The system parent knows where (which commports) the children are connected.
- Each parent and child connection is direct. There are no intermediate devices involved in order for the parent to bootload the child.

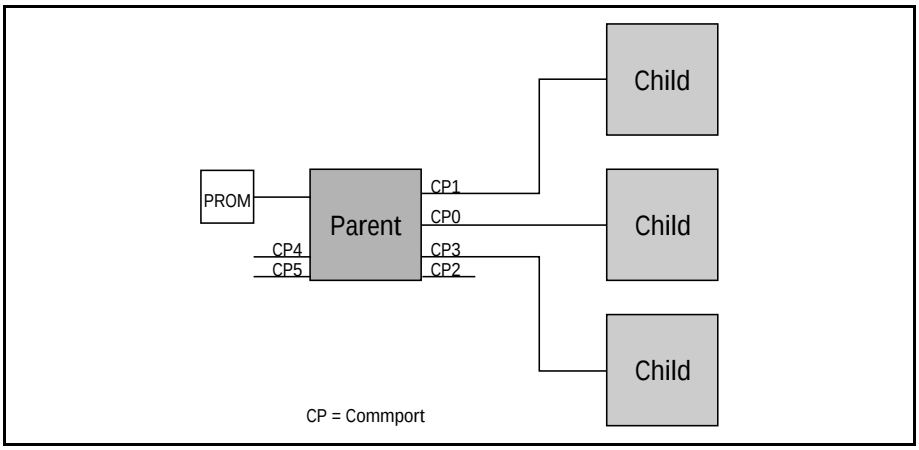

Figure 1. Known, direct connect network

Figure 1 shows the system parent directly connected to the children. At a system reset the parent bootloads from external memory (PROM) and begins performing its tasks as the system parent. Each child polls its commports (function of the on-chip ROM bootloader) to see if the parent is attempting a boot. The parent knows where (which commport) the child is connected. The example C code below demonstrates the process the parent follows in order to bootload the network. Pointers are initialized at the beginning and end of the boot table. The pointer at the beginning of the boot table is incremented after each word is transmitted to the child (out_word command) and continues until the end of the boot table is reached. At this point the child is booted and the parent repeats for all system children.

```
int port_addr = 100040h;                  /* init port pointer to commport 0                        */
extern int  beg_label, end_label;         /* init external labels of the boot table                 */
int port [6] = {1, 2, -1, 3, -1, -1};     /* init CP connection matrix. -1's indicate no connect    */
volatile int *table_ptr, *end_ptr;        /* init the boot table pointer and end table pointer      */
end_ptr = &end_label;                     /* set end pointer to the end of boot table               */
main
{
    for (CPX = 0; CPX  6; CPX++)          /* Continue for all six commports                         */
    {
        if (port [CPX] = 0)               /* If child is connected then boot (!= -1)                */

             table_ptr = &beg_label;      /* Set table ptr to the begining of boot table            */
             port_addr = port_addr +(10h*CPX);  /* Point to next commport address                    */
    do
    {
        *(port_addr + 2) = *table_ptr;    /* Copy data at table_ptr to cp out mem. loc              */
        table_ptr++;                      /* Increment ptr to next word of boot table               */
    }while(table_ptr  end_ptr);           /* Continue until complete boot table is                  */
          }                               /* transmitted through CPX.                               */
        CPX++;                            /* Next commport                                          */
    }
}
```

Figure 2. System parent

When the first data word is sent from the parent, the child locks onto the commport, stores the commport address to register AR3, and continues to receive data until the termination word is received. If the child does not receive the complete boot table in the correct format, the child will never completely boot-up. Please refer to section 13.2 of the TMS320C4x User's Guide to learn more about the boot table requirements.

The parent code in Figure 2 uses external labels to reference the boot table. The external labels are resolved when the boot table is linked to the parent's application code. The code which resides in the parent is shown in Figure 4.

To create the boot table and link it to the parent's application, there are several steps which must be taken to convert the child's application code. The steps are listed below:

- The child application should remain self contained, assembled, and linked as an independent application (`child.out`).
- Use the Hex30 Conversion Utility to convert the `.out` file to a PROM programming file in boot table format. Please read the *Hex30 Utility Addendum*

(Lit# SPRU081) to learn more about creating a boot table (-boot command) using Hex30. An example Hex30 command file is shown in Figure 3:

```
child.out        /* Specify input COFF file                  */
-o  child.hex    /* Specify output filename                  */
-a               /* Convert file to Hex30 ASCII format       */
-memwidth 32     /* Word length of device (32 bits wide)     */
-romwidth 32     /* Specify 32 to convert the complete word  */
-boot            /* Hex30 to construct boot table header     */
:                      :                                     :
:                      :                                     :
```

Figure 3. Hex30 command file

- Now use the HEX2ASM conversion utility (executables and directions available on TMS320 BBS in archived file called hex2asm.exe) to convert the Hex30 programmer file `(child.hex)` to an assembly file `(child.asm).` The HEX2ASM utility extracts the valuable data and creates `.word` xxxxxh for each 32-bit word and saves it in a `.sect` table. The HEX2ASM converter can also include global labels. In Figure 5, the global labels which the parent references are beg_label and end_label.

Note: Also please note that labels can be defined in the final linker `.cmd` file when linking the child boot table to the parent application code. This is accomplished by including the following in the linker `.cmd` file:

```
SECTIONS
{
.child:  {_beg_label = .;
          *(.child)
          _end_label = . -1;  }   RAM 1
```

Figure 4. Defining labels in the linker command file

- The output of the HEX2ASM converter is an assembly file. Link the child`.obj` file with the parent's parent`.obj` file.

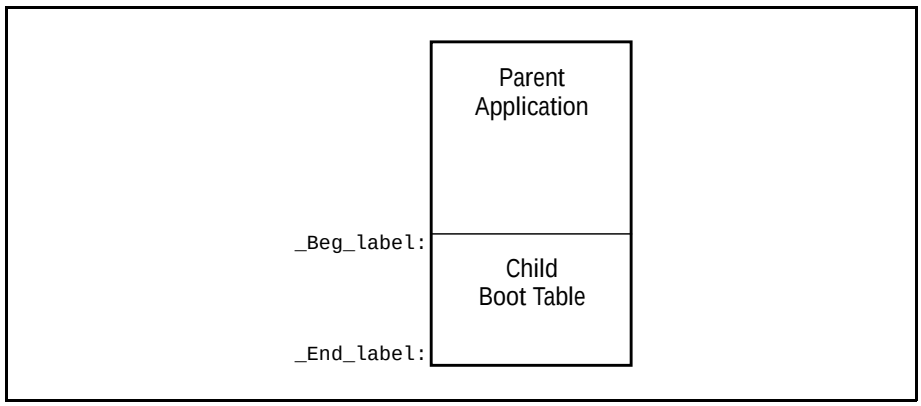

Figure 5. Parent's memory map

At this point the parent application is linked to the child application (in boot table format) as illustrated in Figure 5. At reset, the parent device must bootload this information from external memory or a PC platform. If the parent boots from external memory the final (parent+child) code must be converted again by the Hex30 utility to create the PROM programmer file. If booting the parent via a PC platform, the data must be converted again in order for the PC to read and transmit the boot information properly.