Multiple Flash Manufacturers for a Single Design

Application Note
by Jett Boyd, Field Applications Engineer, Advanced Micro Devices

This application note describes the general method to support different Flash devices in the same circuit. Please note that because of specific hardware and software issues, the final software design will have to be modified and tested in the actual circuit. Refer to the manufacturer’s device data books for more details on Flash programming algorithms. The examples in this paper are not intended to be used for any purpose other than as an aid to understanding the concepts involved.

In the competitive electronics industry, manufacturers of electronic systems often use alternate sources for semiconductor components in order to lower system manufacturing cost and improve delivery schedules. If a particular component is unavailable, an alternate part can be used and prevent manufacturing delays. During the circuit board manufacturing process either part can be soldered to the circuit board. Later, the product software can make adjustments for any firmware differences between the two parts. Semiconductor designers incorporate a manufacturer’s identification number as well as a device identification code into their parts which can be accessed by user software. This allows the software to interrogate the hardware and identify the installed components without having to have a separate software version for each possible hardware build option.

Flash memory devices from AMD, Intel, and other Flash suppliers are often used in the same circuit design. The two areas that need to be investigated before an alternate part is installed are the pinout differences and the software differences. Most manufacturers follow the JEDEC standards for pin placement which makes swapping parts much easier. Any pinout differences may be accounted for by using jumpers to swap signals or, if board space permits, two separate footprints can be placed on the board. Figure 1 shows how two device footprints can be over-lapped to reduce space yet still support different pinouts. In this example pins 1 and 2 are swapped, pins 15 and 9 are swapped, and pins 9 through 14 are shifted by one pin. By changing the component placement during manufacturing, the signals are connected properly. Sometimes there are different programming voltages for different Flash devices. For example, AMD has 5.0 Volt-only Flash devices which are pin-compatible to 12 Volt Flash devices except that the VPP pin is a no connect on the 5.0 Volt-only part. (This pin is also used for an additional address line for higher density parts.) By comparing the pinouts of the two devices, reading the data sheets, and using a little creativity, the circuit designer can usually find a way to solve any hardware differences between two devices.

Figure 1. Board Layout to Support Two Pinouts for One Device

The second issue is how does the software support two Flash devices that use different software algorithms. Some Flash devices require rather involved programming routines with precise timing restrictions. Other parts, such as AMD’s 5.0 Volt-only Flash, use Embedded Algorithms which simplify the software requirements and reduce processor overhead. Many software engineers think that supporting more than one Flash algorithm will make their code much longer. But by comparing the software requirements of two Flash devices and taking advantage of any similarities, the additional code required is usually less than expected. The first step is for the software to access the part and identify which part is installed in the circuit. Once the part is identified, the software can modify the program flow to use the correct algorithm. The remainder of the application note uses Flash devices from AMD and Intel as examples of devices that can be used in the same circuit.

The technique for identifying a component is shown in Figure 2. This example is for the AMD 29F400B and the Intel 28F400BX-B which are both 4 Megabit Flash memories that can be configured as byte-wide or word-wide. The codes shown are for word-wide configuration.
The first step is to read the manufacturer and device code from the device. The sequence to read the AMD manufacturer and device code the sequence is shown in Table 1. The sequence to read the Intel manufacturer and device code the sequence is shown in Table 2. Note that the sequence for AMD can be used for both AMD and Intel because the first two write cycles are ignored by the Intel parts. The following ‘C’ code example shows the routine to read the manufacturer and device code.

Table 1. AMD Autoselect Command

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Address</th>
<th>Data</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5555H</td>
<td>AAH</td>
<td>Write</td>
</tr>
<tr>
<td>2</td>
<td>2AAAAH</td>
<td>55H</td>
<td>Write</td>
</tr>
<tr>
<td>3</td>
<td>5555H</td>
<td>90H</td>
<td>Write</td>
</tr>
<tr>
<td>4</td>
<td>XX00</td>
<td>01H</td>
<td>Read</td>
</tr>
<tr>
<td>5</td>
<td>XX01</td>
<td>22ABH</td>
<td>Read</td>
</tr>
</tbody>
</table>

Table 2. Intel Intelligent Identifier Command

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Address</th>
<th>Data</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>XXXXH</td>
<td>90H</td>
<td>Write</td>
</tr>
<tr>
<td>2</td>
<td>XX00</td>
<td>89H</td>
<td>Read</td>
</tr>
<tr>
<td>3</td>
<td>XX01</td>
<td>4471H</td>
<td>Read</td>
</tr>
</tbody>
</table>
void read_device_type()
{
    unsigned char vendor, device_id;
    unsigned long const AMD_code = 0x1;
    unsigned long const Intel_code = 0x89;
    unsigned long const AM29f400 = 0x22AB;
    unsigned long const i28f400 = 0x4471;
    Flash_write (0x5555,0xaa);    /* Setup 1 */
    Flash_write (0x2aaa,0x55);    /* Setup 2 */
    Flash_write (0x5555,0x90);    /* Setup 3 */
    vendor = Flash_read(0x0);     /* read mfg code */
    device_id = Flash_read(0x1);  /* read device code */
    if (vendor == AMD_code && device_id == AM29f400)
        printf ("The device is a AMD Am29f400./n");
    if (vendor == Intel_code && device_id == i28f400)
        printf ("The device is a Intel i28f400./n");
}

After the device has been identified, the software program can then proceed to use the specific algorithm for the AMD or Intel device for program, erase, or read operations. The AMD and Intel parts use similar commands to initiate read, write, erase operations. These commands are from one to six bus cycles in length. The commands initiate a state machine inside the part that performs the device operation. For example, sending a program setup command and then a program address and program data to the Flash device will cause the data to be written to the specified address. A data polling technique is used to determine when the program operation is complete. Because there are only a handful of commands required, the amount of memory required to support more than one algorithm is very small.

There are two approaches to designing the programming algorithms to support multiple Flash devices. One way is to read the manufacturer and device codes and then branch to one of two parallel routines, one written and optimized for AMD algorithms, the other written and optimized for Intel algorithms. This technique is shown in Figure 3. The other approach is to create one generalized algorithm which contains conditional statements that cause the correct commands to be sent to the Flash device. The conditional statements, such as an IF or CASE, use the manufacturer and device codes to control the program flow. Figure 4 shows this type of program flow. Either approach is valid, although using two parallel routines can make code more readable.
Using Multiple Flash Manufacturers for a Single Design

Figure 3. Parallel Algorithms

Figure 4. In-Line Statements
The following C code examples show how the different algorithms can be combined to program different Flash devices. First we'll look at the algorithm used to program AMD Flash devices. The routine `flash_read()` reads a word from the Flash and the routine `flash_write()` sends a word of data to the specified address. The details of these two sub-routines depend on the actual hardware.

```c
unsigned char write_word_AMD (long address, short word)
{
    unsigned char temp1, temp;
    unsigned char const STATUS_7 = 0x80;
    unsigned char const TIME_OUT_MASK = 0x20;
    long i;
    Flash_write (0x5555,0xaa);   /* Setup 1 */
    Flash_write (0x2aaa,0x55);   /* Setup 2 */
    Flash_write (0x5555,0xa0);   /* Setup 3 */
    Flash_write (address,word);  /* Send address and Data */
    temp1 = word & STATUS_7;    /* Mask off bit 7 */
    for (i=0; i<
        { 
            temp = Flash_read();
        }
    if (((temp & STATUS_7) == temp1)) /* Is bit 7 correct */
    {
        printf("Byte programmed successfully. ");
        return (1);
    }
    }
    if (temp & TIME_OUT_MASK)
        printf(" EXCEEDED PROGRAM TIME LIMIT ");
    return (0);
    printf(" PROGRAMMING IS TAKING TOO LONG ");
    return (0);
}
```
The next example is for the routine to program a byte to the Intel part. The routine flash_read() reads a word from the Flash. The routine flash_write() sends a word of data to the specified address. Remember that these two subroutines must be customized for the actual hardware.

```c
unsigned char write_word (long address, short word)
{
    unsigned char status;
    unsigned char const STATUS_7 = 0x80;
    unsigned char const TIME_OUT_MASK = 0x10;
    unsigned char const VPP_ERROR = 0x08;
    long i;
    Flash_write (address,0x40);  /* Setup 1 */
    Flash_write (address,word);  /* Send address and Data */
    for (i=0; i
    {
        status = Flash_read();
        if (((status & STATUS_7) == STATUS_7))
        {
            printf("Byte programmed successfully. ");
            return (1);
        }
    }
    if (((status & TIME_OUT_MASK) == TIME_OUT_MASK))
    {
        printf(" EXCEEDED PROGRAM TIME LIMIT ");
        return (0);
    }
    if (((status & VPP_ERROR) == VPP_ERROR))
    {
        printf(" Vpp Voltage is OUT of RANGE ");
        return (0);
    }
}
```
Here is an example that uses the manufacturer and device codes to program either part. The reading of the manufacturer and device code is done during initialization earlier in the program.

```c
unsigned char write_word (long address, short word)
{
    unsigned char status, temp, temp1;
    unsigned char const TOGGLE_MASK = 0x40;
    unsigned char const TIME_OUT_MASK2 = 0x20;
    unsigned char const STATUS_MASK = 0x80;
    unsigned char const VPP_ERROR = 0x08;
    long i;
    if (device_type == AMD29F400)
        { Flash_write (0x5555,0xaa);   /* Setup AMD 1 */
            Flash_write (0x2aaa,0x55);   /* Setup AMD 2 */
            Flash_write (0x5555,0xa0);   /* Setup AMD 3 */
        }
    if (device_type == i28F400)
        { Flash_write (address,0x40);   /* Setup Intel */
            Flash_write (address,word);      /* Send address and Data */
            temp1 = word & STATUS_7;
            for (i=0; i<1000;i++)/*Try1000timesthen exit */
                { status = Flash_read(address);     /* read data from Flash */
                    if (device_type == AMD29F400)
                        { if (((status & STATUS_7) == temp1)) /* Is bit 7 correct? */
                            { printf("Byte programmed successfully. ");
                                return (1);
                            }
                        }
                    if (temp & TIME_OUT_MASK1)
                        { printf(" EXCEEDED PROGRAM TIME LIMIT ");
                            return (0);
                        }
                    }
                }
        }
    if (device_type == i28F400)
        { if (((status & STATUS_7))
            { printf("Byte programmed successfully. ");
                return (1);
            }
        }
    if (((status & TIME_OUT_MASK2) == TIME_OUT_MASK2))
        { printf(" EXCEEDED PROGRAM TIME LIMIT ");
            return (0);
        }
    if (((status & VPP_ERROR) == VPP_ERROR))
        { printf(" Vpp Voltage is OUT of RANGE ");
            return (0);
        }
    printf(" PROGRAMMING IS TAKING TOO LONG ");
    return (0);
}
```
Summary
While these examples are just part of the software required to interface to Flash devices, they help illustrate how easy it is to support more than one Flash manufacturer in a single design. By comparing the command set for two devices, the software designer can optimize the Flash interface routines and take advantage of any similarity of the commands. By making small adjustments in the hardware and software, it is comparatively easy to use multiple Flash devices in a single design.