

Introduction

The HMP8154 and HMP8156A are NTSC/PAL video encoders. The difference between the two parts is that the HMP8154 provides an optional 3-line vertical flicker filter while the HMP8156A does not. In all other ways, including their read-only product ID registers, the parts are the same.

The product ID register is an internal register which may be read via the encoder's I2C bus. Reading the product ID register allows an application to determine the system's hardware configuration. For most applications, only one of the HMP8156A or HMP8154 is used and it is not an issue that both parts return the same product ID register value.

However, applications which are required to uniquely identify both encoders must use additional steps to determine which part is responding. This application note outlines a method to uniquely identify the HMP8154 and HMP8156A via the I2C bus.

Procedure Outline

The first step is to read the product ID register via the I2C bus. The slave address for the encoder is selectable using the SA input pin. The address is 0x40 when the pin is low or 0x42 when the pin is high. The subaddress for the product ID register is 0. If the value read is 0x54, then either an HMP8156A or HMP8154 has responded. Determining which one uses bits of the video processing register.

The video processing register is located at subaddress 3. The 0x02 bit in the register controls the flicker filter in the HMP8154. In the HMP8156A, this bit is a read only bit which always returns a zero when read. The HMP8156A ignores all writes to the 0x02 bit of the video processing register.

By writing a one to the 0x02 bit of the video processing register and then reading the value back, the parts can be identified. The HMP8154 will accept the write and return a one when read. The HMP8156A will ignore the write and return a zero when read.

C Programming Language Code Example

The procedure outlined above is implemented in the C function `getEncoderID()` shown in Figure 1. The function reads the product ID register value from the encoder. If an HMP8156A is detected, then the value 0x56A is returned. Otherwise, the product ID value is returned unchanged. Example product ID values from other Intersil encoders are 0x54, 0x56, and 0x71.

The function requires one argument which indicates the state of the encoder's I2C address select input, SA. The input selects slave address 0x40 or 0x42. These are the assigned addresses for Intersil encoders.

The function calls two other, externally defined, functions to read and write I2C registers: `readI2CReg()` and `writI2CReg()`. These functions and their related hardware drivers must be provided by the user.

As the declarations show, the functions `readI2CReg()` and `writI2CReg()` have similar arguments: `int slvAdrs` and `int regAdrs`. The first argument contains the I2C bus address of the slave device targeted. The second argument, `regAdrs`, contains the subaddress of the register being accessed within the target device. The function `readI2CReg()` returns the value read as an integer while `writI2CReg()` has no return value. Instead, it requires a third argument, `int value`, which contains the data to be written to the specified register.

The details of the functions to read and write the I2C bus are very application specific so examples are not included here. Typically, the functions only need to control the data and clock signals of the I2C bus, SCL and SDA. More details about I2C bus read and write cycles are given in the HMP8154 data sheet, FN4343.1.

The example code was written using standard ANSI C syntax and has been compiled. The circled numbers are references to the explanations.

Summary

This Tech Brief has described a method which may be employed to uniquely identify the HMP8154 and HMP8156A encoders via their I2C interface. The method is not complicated and adds only two to four additional I2C accesses.

```

extern int readI2CReg(int iSlvAdrs,
                    int iRegAdrs);
extern void writI2CReg(int iSlvAdrs,
                    int iRegAdrs, int iValue);

#define PRODID      0
#define VIDPROC     3
#define FLICKEN     2

int
getEncoderID(int iAdrsSel)
{
    int    iSlvAdrs,
          iProdID,
          iOrigVidProc,
          iFlickVidProc,
          iVidProcIn;

    iSlvAdrs = 0x40 | (iAdrsSel ? 2 : 0);

    iProdID = readI2CReg(iSlvAdrs, PRODID);
    if (iProdID == 0x54)
    {
        iOrigVidProc = readI2CReg(iSlvAdrs,
                                VIDPROC);

        iFlickVidProc = iOrigVidProc | FLICKEN;
        writI2CReg(iSlvAdrs, VIDPROC,
                  iFlickVidProc);

        iVidProcIn = readI2CReg(iSlvAdrs,
                                VIDPROC);

        if (iVidProcIn & FLICKEN == 0)
        {
            iProdID = 0x56A;
        }

        writI2CReg(iSlvAdrs, VIDPROC,
                  iOrigVidProc);
    }

    return iProdID;
}

```

FIGURE 1. GET ENCODER PRODUCT ID FUNCTION

1. Declare the external functions called by `getEncoderID()`. These are described in more detail above.
2. Define compiler macros for the sub-addresses of the product ID and video processing registers and the bit mask for the flicker filter enable bit.
3. Define the `getEncoderID()` function. Its integer argument is the slave address select flag described in (5) below. The function returns the integer product ID in (12).
4. Define the internal variables used for the slave address of the part, the value of the product ID register, the original value of the video processing register, the same value with the flicker filter enable bit set, and the value read after the bit has been written.
5. Assign the slave address on the I2C bus where the encoder is expected. If the slave address select flag is true, then the function uses slave address 0x42. Otherwise, it uses slave address 0x40. The SA input pin performs the same function in the hardware.
6. Read and save the product ID register so that it may be checked. If the value read is 0x54, then either the HMP8154 or the HMP8156A is present and (7-11) are required to determine which one. Otherwise, another encoder, the HMP8156 or HMP8173 for example, is present. In this case, no further steps are needed and the function returns the product ID in (12).
7. Read the current value of the video processing register and save it so that the register can be restored to its original value. (This assumes that no other processes are also modifying the encoder registers.) This step is optional if the encoder will be reset or the video processing register will be initialized later.
8. Set the flicker filter enable bit by or-ing it with the video processing register value and writing the modified value to the part.
9. Reread and test the video processing register to determine if the write to the flicker filter enable bit was ignored.
10. The write was ignored since the bit was read back as a zero instead of a one. Adjust the product ID register to indicate an HMP8156A is present.
11. Restore the original value of the video processing register which may have been modified by the write/read/compare steps. This step is optional per (7).
12. Return the product ID value. The value returned is the one read from the hardware (6) or assigned (10).

All Intersil products are manufactured, assembled and tested utilizing ISO9000 quality systems. Intersil Corporation's quality certifications can be viewed at www.intersil.com/design/quality

Intersil products are sold by description only. Intersil Corporation reserves the right to make changes in circuit design, software and/or specifications at any time without notice. Accordingly, the reader is cautioned to verify that data sheets are current before placing orders. Information furnished by Intersil is believed to be accurate and reliable. However, no responsibility is assumed by Intersil or its subsidiaries for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Intersil or its subsidiaries.

For information regarding Intersil Corporation and its products, see www.intersil.com