

Using the AD7732/AD7734/AD7738/AD7739 Checksum Register

by Tom Meany

This application note refers solely to the AD7738 but is fully applicable to the AD7732, AD7734, and AD7739.

The AD7738 features a powerful digital interface with configuration flexibility. However, this means that the user may have to write tens of bytes of information to correctly configure the ADC. In the noisy electrical environment where these devices have to operate, bytes can be easily corrupted when being sent to or read from the ADC. Typically, users write a byte and then read it back to confirm that it has been correctly received. When reading conversions, it is common for users to read the data register three times and compare the three results to validate the data. This application note explains the use of the AD7738's checksum register to verify data integrity.

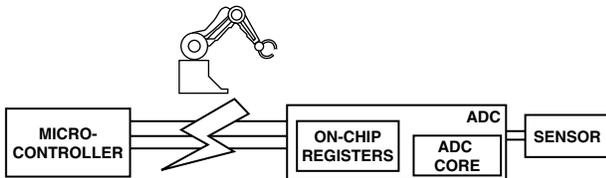


Figure 1. An ADC in a Noisy Environment

The checksum register is a 16-bit register and is read by writing 0x45 to the communications register. After a 32 ones reset or a reset using the RESET pin, the register should read back 0xE45E. If the register is read again, it will contain 0x4691 and again reads 0x9AEB before the sequence eventually repeats after 65,535 reads. What is going on?

The checksum register is actually an LFSR (linear feedback shift register). In fact, it's a maximal length LFSR, which is why it takes 65,535 cycles before the data repeats.

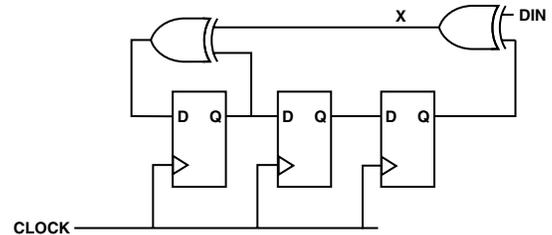


Figure 2. Simple LFSR Circuit Showing a 3-Bit LFSR. (AD7738 contains a 16-bit LFSR. See the C code for exact feedback.)

For the AD7738 checksum register, there are 16 flip-flops, and the feedback is from the 15th bit to Bits 0, 1, 2, and 5. The feedback positions for a given length LFSR are generally read from standard tables such as those found in the references.

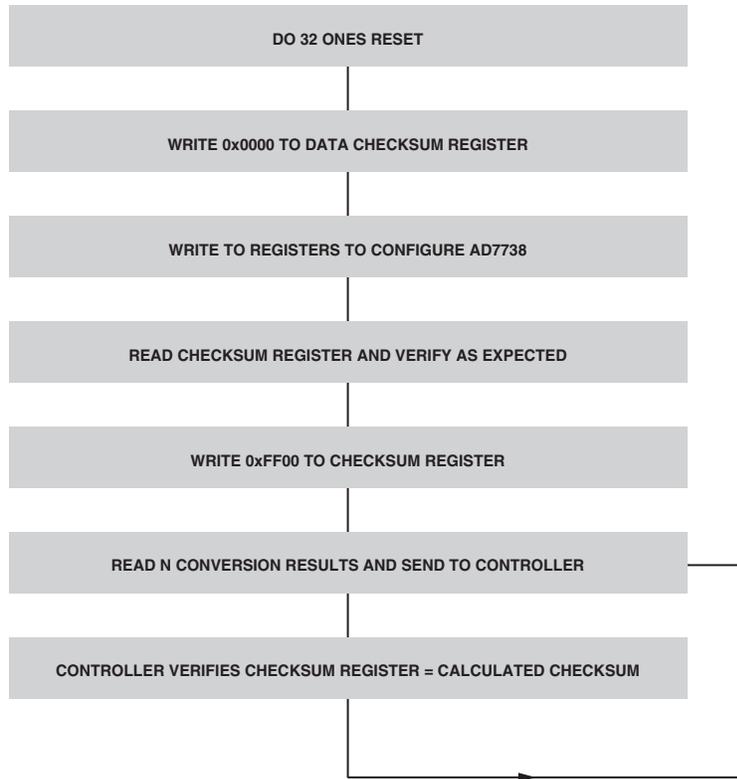
On power-up, the checksum register input is connected to the DIN pin and the clock to the SCLK pin. Each time a bit of data is written to the AD7738, the content of the LFSR updates in a predictable way, depending on whether that bit of data is a 0 or a 1. In this way, the interface data written to the AD7738 changes the contents of the checksum register. A system controller can use this feature to confirm that the data written to an AD7738 has been correctly received. For instance, writing all eight sets of channel calibration registers would involve writing $8 \times 2 \times 3 = 48$ bytes of data to the AD7738. Without a checksum register, the user would have to read back all 48 bytes to confirm that the data was correctly received. With the checksum register, the user can precompute an expected value for the checksum register and confirm that the 48 bytes arrived uncorrupted by reading back just two bytes from the checksum register.

Similarly the checksum register can monitor DOUT. If reading back ADC conversion results, the system controller could read the checksum register every 10, 100, or 1000 conversions to confirm that the contents of the checksum register match the expected value, where the expected value is updated on every data transfer by a simulated checksum register within the controller.

AN-626

Writing 0x0000 to the compression register resets the checksum register to 0xFFFF and sets it to monitor DIN. If 0xFF00 is written to the checksum register, then the checksum register is reset to 0xFFFF and it monitors DOUT.

If 0x0000 is written to the checksum register and the checksum register is then read, the user will read back 0xE45E because the write to the communications register to select the read of the checksum register will change the contents of the register (in a predictable way). However, if 0xFF00 is written, then 0xFFFF will be read back since there is no read between setting up the checksum register and reading its contents.



When used to check the configuration data written to the ADC, the expected checksum contents can be precalculated using the code shown below. However, when used

to check data conversion results, the routine below will need to be used to calculate the expected value in real time. The code can easily be coded in assembly language on almost any microcontroller.

```
// Reset the register to all ones
for (i=0;i<=15;i++) q[i]=1;

// Read in one bit at a time and update the register's flip-flops
j = 1;
while (1){
    scanf("%d",&data);

    if (data==q[15]) qq[0]=0; // XOR
    else qq[0]=1;

    qq[1]=q[0];

    if (q[1]==q[15]) qq[2]=0; // XOR
    else qq[2]=1;

    if (q[2]==q[15]) qq[3]=0; // XOR
    else qq[3]=1;

    qq[4]=q[3];

    if (q[4]==q[15]) qq[5]=0; // XOR
    else qq[5]=1;

    qq[6]=q[5];qq[7]=q[6];qq[8]=q[7];qq[9]=q[8];
    qq[10]=q[9];qq[11]=q[10];qq[12]=q[11];qq[13]=q[12];
    qq[14]=q[13];qq[15]=q[14];

    printf("%8d  ",j);
    for (i=15;i>=0;i--){
        q[i]=qq[i];
        printf("%1d",q[i]);
    }
    printf("\n");

    if ((j%8)==0) printf("-----\n");
    j++;
}
```

This code can be implemented in real time if monitoring DOUT and a sufficiently fast processor is available. However, when monitoring DIN, it is generally the case that the data to be written to the part is known in advance. In this case, the C code previously shown can be used to precompute the expected checksum value and the controller reads this value from memory when a comparison is required. Similarly, the expected value could be gathered by reading the checksum register under ideal conditions and using this value as the expected value thereafter. If implementing the preceding code, an interesting check is to input four all-zero bytes (four writes to the communications register \geq no effect) and then enter 0x45 (telling the AD7738 that the next operation is a read of the checksum register) and the checksum register contents should be 0xABCD.

The checksum register is an application of signature analysis or data compression. One concern with data compression is data aliasing. With a 16-bit register, there are 65,536 possible combinations. Therefore, even if the incoming data is wrong, there is a 1/65,535 chance that the data in the register will be correct. A 24-bit register would have a much lower occurrence of aliasing, but the larger amount of data to be read would outweigh the benefits.

REFERENCES

Maxfield, Clive. *Bebop to the Boolean Boogie*, Butterworth-Heinemann, 2nd ed., 2002.

"The Ouroboros of the Digital Consciousness: Linear-Feedback Shift Registers." EDN, January 4, 1996.