

Introduction

Automatic gain control (AGC) is used in many applications. In receivers for example, the in-coming signal amplitude is often controlled using an AGC in order to make maximum use of the dynamic range of the Analog to Digital Converter (ADC) that follows.

This application note describes 2 very different methods of creating an AGC circuit using the AN221E04 Field Programmable Analog Array (FPAA). The first uses a simple circuit with the gain controlled by an external microprocessor. This method offers the advantages of speed of response and the ability to also control other parameters such as the signal's offset. The second uses a more complicated circuit but has the advantage of being self-contained i.e. it does not require microprocessor control.

Method 1: Microprocessor Control

Figure 1 shows a typical circuit used for AGC. The heart of the circuit is a simple gain stage (GainInv CAM) whose gain is continually adjusted by the microprocessor. A signal is input on I3P and I3N at the lower left. The amplitude adjusted signal is output on O1P and O1N at the upper right.

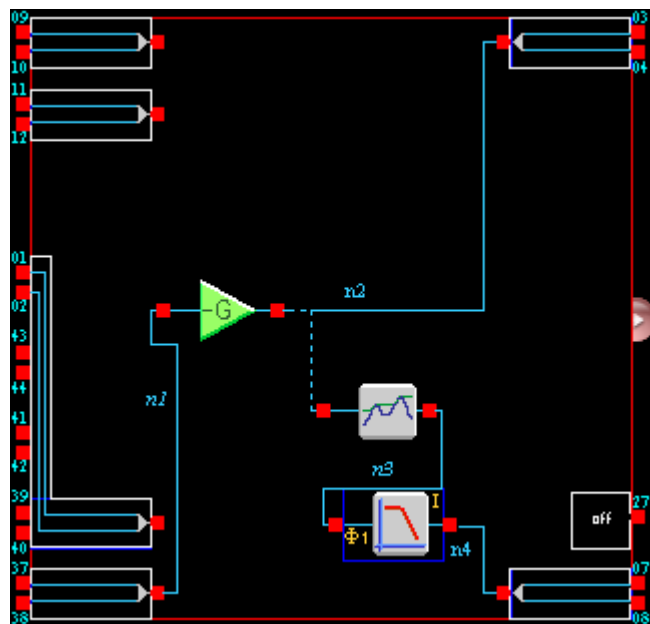


Figure 1. uP Controlled AGC Circuit

The rest of the circuit consists of a peak detector and a low pass filter. This optional part of the circuit is designed to convert the signal amplitude to a DC level for output to an ADC. The advantage of this circuitry is that it makes it easy to measure the signal amplitude with just one measurement. The disadvantage is that the

low pass filters will slow the response of the DC level to changes in signal amplitude. Care must be taken to choose a corner frequency for the filter that is low enough to remove ripple from the DC level, but high enough to provide adequate speed of response. The appendix shows some example code for an AGC using this circuit (this code should be used in conjunction with the code generated by AnadigmDesigner2). Note that an alternative using a peak detector is to make a number of direct measurements of the output signal and calculate the amplitude in software. This makes the circuit much simpler but the sampling of the signal slower. Figure 2 shows the input signal with modulated amplitude, and the output signal with controlled amplitude.

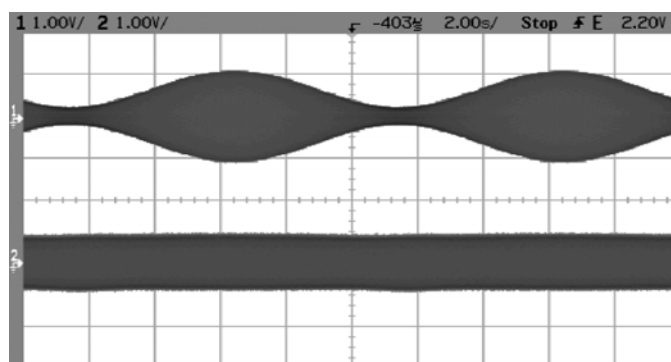


Figure 2. Operation of the AGC circuit

One advantage of using microprocessor control is that it can react quickly to changes in signal amplitude. The time taken for a reconfiguration of the AN221E04 can be less than 4us if clocked at the maximum rate of 40MHz. The gain of the GainInvCAM will then respond immediately at the end of the reconfiguration with no signal interruption. The waveform in Figure 3 shows the effect of reconfiguring a large increase in gain.

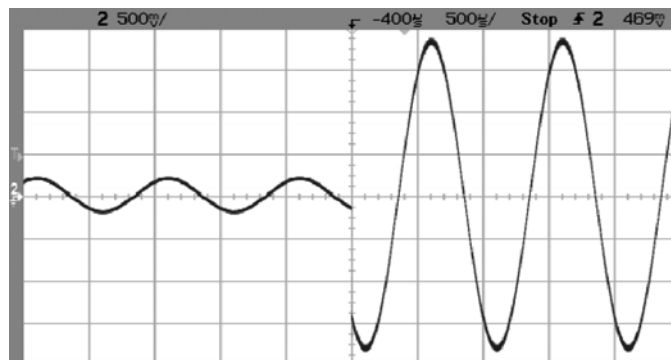


Figure 3. Step Response in Gain

Figure 4 shows the step in signal amplitude in more detail. It reveals that the signal has reached it's new level in less than 0.5us. Note that to achieve this

response, the GainInv CAM was clocked with a 16MHz analog clock and output filtering was not used.

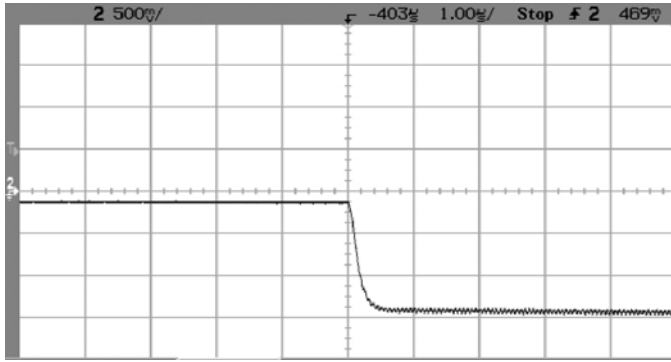


Figure 4. Step Response in Gain is <0.5us

Offset & Amplitude Control

Any CAM parameter can be controlled from a microprocessor. Figure 5 shows another AGC circuit for control of both amplitude and offset. A signal is input on I3P and I3N at the lower left. This time the heart of the circuit consists of a 3 input summing gain stage (SumInv CAM). The signal is connected to the upper input of the SumInv CAM, a +3V reference is connected to the middle input, and a -3V reference is connected to the lower input. The gain on the upper input is modified by the microprocessor to control the signal amplitude. The gain on the lower 2 inputs is modified to control the signal offset.

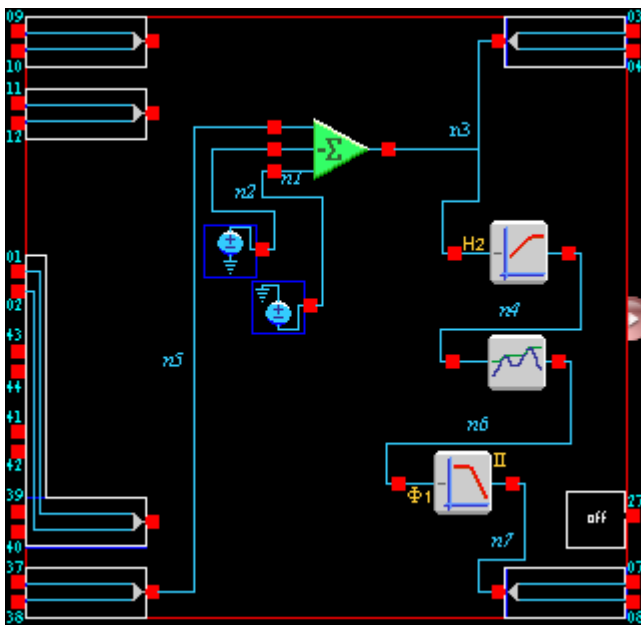


Figure 5. Amplitude & Offset Control

The rest of the circuit in Figure 5 is optional and is used to convert amplitude and offset into DC levels. The signal is connected to a high pass filter which removes any DC level. This signal is then fed through a peak detector and low pass filtered to provide a DC level which is proportional to the signal amplitude (as before). After measuring the amplitude, the circuit can be

reconfigured by the microprocessor to bypass the highpass filter and peak detector, and to set the corner frequency of the lowpass filter to a very low value. This would provide a DC level which is proportional to the signal offset. In this way, both amplitude and offset can be measured. Alternatively, a number of direct measurements can be made of the output signal and amplitude/offset calculated in software.

Method 2: Voltage Controlled Gain

The second method has the advantage that the circuit is self-contained i.e. it doesn't require microprocessor control. The circuit is shown in Figure 6.

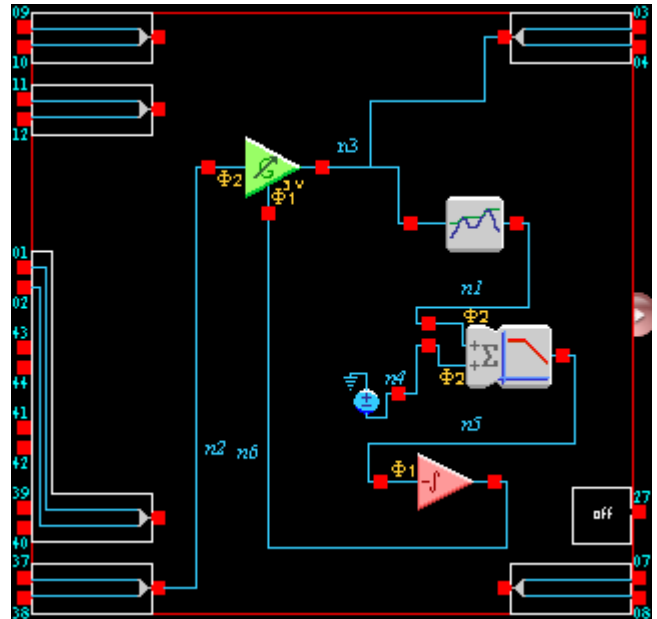


Figure 6. AGC Circuit without uP Control

This circuit works by using negative feedback with a voltage controlled gain stage (VCG CAM). The signal out of the VCG CAM is connected to a peak detector to provide a DC level proportional to the signal amplitude. This level is then offset and smoothed using a 2 input SumFilter CAM. The offset should be such that the output of the SumFilter CAM is 0V at the required signal amplitude (i.e. when the VCG output is at the desired amplitude). The output of the SumFilter CAM is connected to an inverting integrator whose output is used to control the gain of the VCG CAM. If the signal amplitude increases above the required level, a positive voltage will appear on the input of the inverting integrator causing its output to ramp down. This will lower the gain of the VCG CAM until the correct amplitude is achieved. If the signal amplitude falls below the required level, a negative voltage will appear on the input of the inverting integrator causing its output to ramp up so increasing the gain of the VCG CAM.

In the above circuit, the VCG CAM should be loaded with a gain profile that increases with increasing control voltage. Whether this profile is linear or not is up to the user.

Appendix - Example Code for Microprocessor Controlled AGC

NOTE: Anadigm can supply example code for an AGC application on their own development boards. Owners of development boards can obtain example code by emailing Anadigm support. Example code will be written using Visual C++ v6.

```

////////////////////////////////////
// AGC.c : Demonstration of Automatic Gain Control
//
// Author: Dave Lovell 26/7/02
// Tested: Dave Lovell 26/7/02
// Notes: This software should be used in conjunction with the software
//        files generated by AnadigmDesigner2 for the circuit in Figure 1.
//        These files have been given the default names ApiCode.c, ApiCode.h,
//        CAMCode.c and CAMCode.h.
//        This software was tested using an ADC + 3v9 zener reference mounted
//        onto the breadboard area of the development board.
////////////////////////////////////

#include <stdio.h>
#include <conio.h>
#include "ApiCode.h"
#include "CAMCode.h"
#include "AGC.h"

const an_Byte StartPadBytes[] = {0x0,0x0,0x0,0x0,0x0};
const an_Byte EndPadBytes[] = {0x0};

/*****
 * Procedure Name : PrimaryConfigChip
 *
 * Purpose       : This function calls the necessary C Code API functions to
 *                 retrieve and send off the primary configuration for this
 *                 application.
 *
 * Inputs        : unsigned char chip_id - Which chip to configure.
 *
 * Outputs       : None, but the AN220 will be configured according to the
 *                 Primary Configuration data stream.
 *****/
void PrimaryConfigChip(an_Byte chip_id)
{
    int StreamSize;                //set up variable for size of config stream
    const an_Byte* PriConfigData;  //set up pointer to point to pri config stream

    PriConfigData = an_GetPrimaryConfigData(chip_id, &StreamSize); //go get stream size & location
    Configure(StartPadBytes,4); //start with 5 pad bytes
    Configure(PriConfigData, StreamSize); //go configure, pass stream size & stream pointer
    Configure(EndPadBytes,0); //finish with 1 pad byte
}

/*****
 * Procedure Name : Configure
 *
 * Purpose       : This function sends the configuration data to the chip.
 *                 Note that the core of this function has been omitted
 *                 because the precise implementation is system dependent.
 *
 * Inputs        : unsigned char* data - Pointer to the config data stream.
 *                 int size - Size of config data stream.
 *
 * Outputs       : None, but the AN220E04 will be configured
 *****/
void Configure(const an_Byte* data, int size)
{
    int i,data_byte;
    for (i=0;i<=size;i++) //i incs through bytes of stream
    {
        data_byte = *(data+i); //get byte from data stream

        //insert code here to send data serially to AN220E04
    }
}

/*****
 * Procedure Name : main
 *
 * Purpose       : This is the main function of the program. It loops forever*
 *                 measuring the signal and adjusting the gain accordingly.
 *
 * Inputs        : None.
 *****/

```

```
* Outputs      : None. The function never returns.      *
*****/
void main()
{
    double SetAmpl = 2;           //required amplitude = 2V
    double Gain = 1;             //starting value of gain = 1
    double Vzener = 3.9;        //zener voltage used as ADC reference
    int ADCOutput;              //ADC output (0 - 255)
    double MeasAmpl;            //measured amplitude (V)
    int StreamSize;              //size of reconfig data stream
    const an_Byte* ReconfigData; //set up a pointer to point to reconfig stream

    PrimaryConfigChip(an_chipl); //do primary configuration

    while (1)                    //loops forever, insert stop condition in brackets
    {
        ADCOutput = ReadADC();    //read ADC
        MeasAmpl = Vzener * ADCOutput / 255; //convert to volts

        Gain = Gain + 0.1 * (SetAmpl - MeasAmpl); //adjust gain by small amount
        if (Gain < 0.01)
            Gain = 0.01; //bottom limit of gain is 0.01
        if (Gain > 100)
            Gain = 100; //top limit of gain is 100

        an_InitializeReconfigData(an_chipl); //allocate memory for reconfig data
        an_setGainInv(an_chipl_GainInv, Gain); //go calculate cap values
        ReconfigData = an_GetReconfigData(an_chipl, &StreamSize); //go get stream size & location
        Configure(ReconfigData, StreamSize); //go configure, pass stream size & ptr to data
        an_ShutdownReconfigData(an_chipl); //free memory allocated to reconfig data
    } ;
}

/*****
* Procedure Name : ReadADC
*
* Purpose      : This function reads the ADC. Code implementation is system dependent.
*
* Inputs       : None.
*
* Outputs      : Returns ADC output (0-255).
*****/
int ReadADC()
{
    int ADCOutput;

    //insert code here to read ADC output

    return ADCOutput;
}
```



For more information logon to: www.anadigm.com