

Analog-to-Digital Converter (ADC 0831) ATE Testing

Test function 1: Continuity

Continuity testing was performed on 7 out of 8 pins (except Vss or gnd) of the ADC 0831 chip. To perform continuity testing, the resources in the DIB must be isolated so that each resource can detect a single pin on the chip. According to the ADC daughter card schematic shown in Figure 1, Control Bits (CBITs) 41, 42, 45, and 46 must be active to trigger the contactors.

The final connections achieved through flipping the contactors would be:

Chip-select (CS) → IO25 (Digital Pin 25)

Vin + (+IN) → (Analog Pin Unit) APU16

Vin - (-IN) → IO24 (Digital Pin 24)

Vcc → APU23

CLK → IO27 (Digital Pin 27)

Dout → IO8 (Digital Pin 8)

VREF → IO26 (Digital Pin 26)

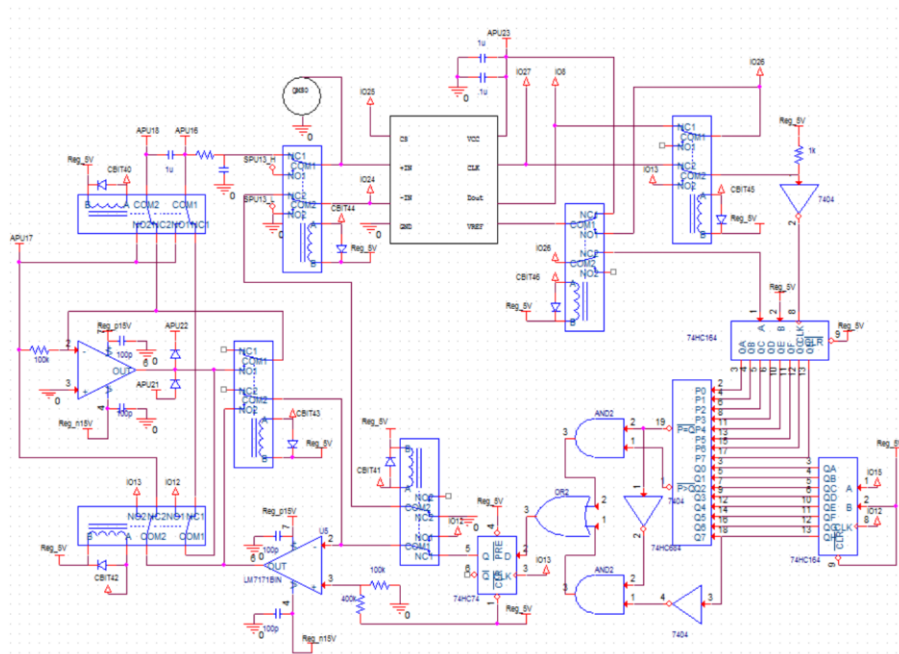


Figure 1: ADC Daughter card Schematic

It may not be ideal to connect digital pins to the chip's analog pins, such as VREF or Input. The digital pin unit (DPU) can also be used as a source measurement unit (such as APU) for continuity purposes, as it can also force voltage or current, depending on the testing need.

Test Procedure:

The test was executed based on whether pins on each chip was connected to an APU or a DPU. DPUs were all grouped in a single trigger that all of them can be enabled, disabled, or forced to current, etc. Thus, the plan was to test all the ADC pins using DPUs, then move on the ones connected to APUs. This was also possible because VDD pin was using an APU, which generally requires grounding when testing continuity for input and output pins.

1. Set APU 23 (Vcc) to force voltage of 0V (grounded),
2. Set APU 16 (IN+) to be floating
3. Wait for settlement
4. Force 1mA of current from all DPU and measure the voltage
5. Log the results as continuity for CS, Vin-, CLK, DOUT, VREF, respectively
6. Set APU 23 and all DPU to force 0V (Ground)
7. Wait for settlement
8. Force 1mA of current and measure the voltage from APU 16
9. Log the data as continuity for Vin +
10. Ground all the input pins for the ADC chip (Vin+, Vin-, CS, CLK, VREF).
11. Wait for settlement
12. Force 1mA of current and measure the voltage from APU 23.
13. Log the measured data as continuity for Vcc
14. Clean Up Code

All pins for continuity tests fell within the specification range, with the Vcc pin barely sitting on the edge of the spec. Testing code is attached in a separate page

Test function 2: Zero-Scale Error (ZSE) & Full-Scale Error (FSE)

Unlike in continuity testing, not all pins required an independent resource from the test equipment, as the datasheet specified that some of the ADC pins could be at the same potential, such as $VCC = VREF = 5V$. CBIT 28, 44, 45 were activated for this test. CBIT 28 activates QMS0 for $Vin+$ to obtain precise input measurements and calculate the ZSE and FSE. CBIT 44 grounds the negative terminal ($Vin -$) of two differential inputs; that way, only the positive terminal ($Vin +$) requires altering for testing purposes. CBIT 45 ties VCC and $VREF$ at the same potential, which is APU 23.

The final connections achieved through flipping the contactors would be:

Chip-select (CS) → IO25 (Digital Pin 25)

$Vin + (+IN)$ → SPU13

$Vin - (-IN)$ → Ground

Vcc → APU23

CLK → IO27 (Digital Pin 27)

Dout → IO8 (Digital Pin 8)

$VREF$ → APU23

Timing:

The Analog inputs can be consistently driven to whatever voltage at any point to achieve a desired output. The complex part of this test is how to capture the output data that is digital, meaning many 1s and 0s, rather than an analog voltage that can easily be measured. The digital output data must be captured, and that can only be achieved through proper timing and sequence of CS, CLK, and Dout Pin.

The timing diagram for capturing the digital output of ADC 0831 is shown in Figure 2. In this project, the data output timing (Figure 16) and the start conversion timing diagram (Figure 17) are shown. The digital vector editors for the corresponding DPUs must be set to proper modes with reasonable clock edges to mimic the correct timing behavior. Following the datasheet, the frequency of all digital pins was scaled to be 250kHz, CS (DPU 25) and CLK (DPU 27) were set as inputs, and Dout (DPU 8) was set as an output.

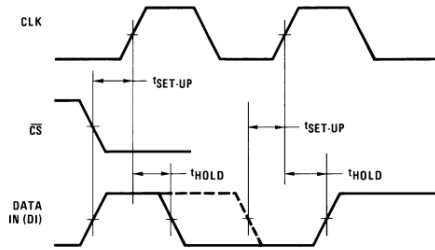


Figure 15. Data Input Timing

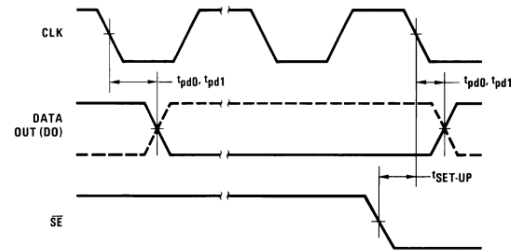


Figure 16. Data Output Timing

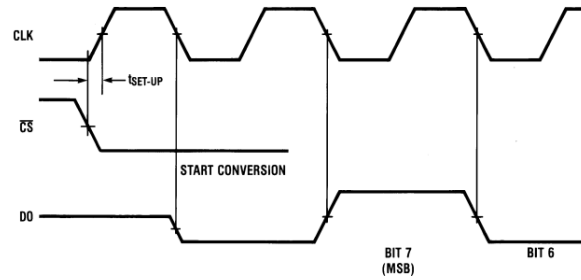


Figure 17. ADC0831-N Start Conversion Timing

Figure 2: Timing Diagrams of ADC 0831.

The CLK was set to a regular clock format with positive and negative edges at 10% and 60% of the period, respectively. The datasheet specified a 40%-60% clock duty cycle as ideal.

The CS was set to return to zero format with positive and negative edges at 30% and 80% of the period, respectively. This allows the next positive edge of the CLK to arrive in the cycle after the minimum setup time (t_{setup}) has passed, which is 250 ns.

$$Period = \frac{1}{f_{clk}} = 4000ns,$$

$$10\% \text{ of the period} = 400ns$$

With the equation above, 80% negative edge of the CS to 10% positive edge of the CLK should be 30% of the period (1200ns), which is greater than the minimum setup time of 250ns.

The Dout is an output, but it does have clock edges, which are receiving edges. These edges determine when the data should be captured. The ADC starts the conversion after the proper setup sequence above, then after a 1 full clock cycle, the data is ready to be captured. For this operation, the edge that primarily matters is the positive receiving edge. Because this edge must account for the falling edge of the clock to the data valid time (T_{pd1}, T_{pd0}), which is 650ns typical. The positive receiving edge of 80% did the job by providing a 20% gap (800ns) between the negative edge of the clock and the positive receiving edge, accounting for some headroom.

Test Procedure:

ZSE: distance of the first edge (the voltage that changes Dout from 0000 0000 to 0000 0001) to half LSB.

FSE: distance of the last edge (the voltage that changes Dout from 1111 1110 to 1111 1111) to $5 - 1\frac{1}{2}$ LSB.

According to the datasheet, the maximum ZSE and FSE were both 1 LSB. Since the device being tested is an 8-bit ADC, the LSB is $5/255 = 19.6$ mV. This test was executed using the sweep method of certain analog voltage ranges. Considering the worst case for both errors, the sweep ranges for ZSE and FSE were calculated below.

$$ZSE \text{ sweep range : } \min = \frac{1}{2}LSB - 1LSB, \max = \frac{1}{2}LSB + 1LSB$$

$$FSE \text{ sweep range: } \min = 5 - \frac{1}{2}LSB - 1LSB, \max = 5 - \frac{1}{2}LSB + 1LSB$$

The first and last edges were measured with a QMS for precision, and the distances between the measured values and the ideal edges were calculated for data logging.

Code snippets are attached to a separate page.

Test function 3: Differential Non-Linearity (DNL) & Integral Non-Linearity (INL)

This test required the same setup as the ZSE and FSE, except it did not require a QMS. A histogram method was used instead of a linear sweep, in which the number of code hits for each Dout value (0 to 255) was stored in a 256-element array. However, to achieve this, a linear sweep from the zero scale to full scale for analog inputs is necessary. As the tester sweeps the input, the digital output value remains constant until it hits the next edge, and the number of times this consistent value appears (code-hits) is stored in a histogram array. Because there are 256 Dout values, the array size is 256. Using this histogram array, the code width of each Digital output can be determined.

Test/Calculation Procedure:

$$\sum_{i=0}^{255} \text{histogram}[i] = \text{total code hits}$$

$$\frac{\text{total code hits}}{256} = \text{codehits}_{avg}$$

$$\text{codewidth}[i] = \frac{\text{histogram}[i]}{\text{codehits}_{avg}}$$

Where the code-width array is constructed using a loop. using a code-width array, an array of DNL, and an INL for each digital output can be constructed via:

$$\text{DNL}[i] = \text{codewidth}[i] - 1$$

$$\text{INL}[i] = \sum_{k=1}^{i-1} \text{DNL}[k]$$

$$\text{INL}[i] = \text{INL}[i - 1] + \text{DNL}[i - 1]$$

For test time optimization, only two loops were used: one to construct a histogram array and another to create the DNL and INL arrays.

Repeatability:

All three test functions were run simultaneously 30 times and logged as a CSV file. The output CSV file was input into a Matlab script, and the results are shown in Figure 3 below. The column to focus here is the repeatability (GRR). A good repeatability standard is GRR under 10%. Histogram plot of test results for individual data logged values are shown in Figure 4.

Parameter	μ	σ	Cp	Cpk	GRR(%)
CS	0.6709	0.000396	336.860768	277.152197	0.296859
IN-	0.792733	0.000512	260.377822	134.919108	0.384057
CLK	0.801933	0.000249	534.522484	264.677717	0.187083
Dout	0.7232	0.0004	333.333333	230.666667	0.3
VREF	0.801	0.000258	516.397779	256.907895	0.193649
IN+	0.711267	0.000442	301.511345	217.640939	0.331662
VCC	-0.2039	0.0003	444.444444	4.333333	0.225
Vos	0.1152	0.064277	5.185871	4.588459	19.283164
AvErr	0.034133	0.027381	12.173921	11.758385	8.21428
DNL	-0.552033	0.066233	5.032764	2.25451	19.869799
INL	-0.644433	0.068077	4.896398	1.740996	20.423176

Figure 3: Matlab Calculation Output for Repeatability

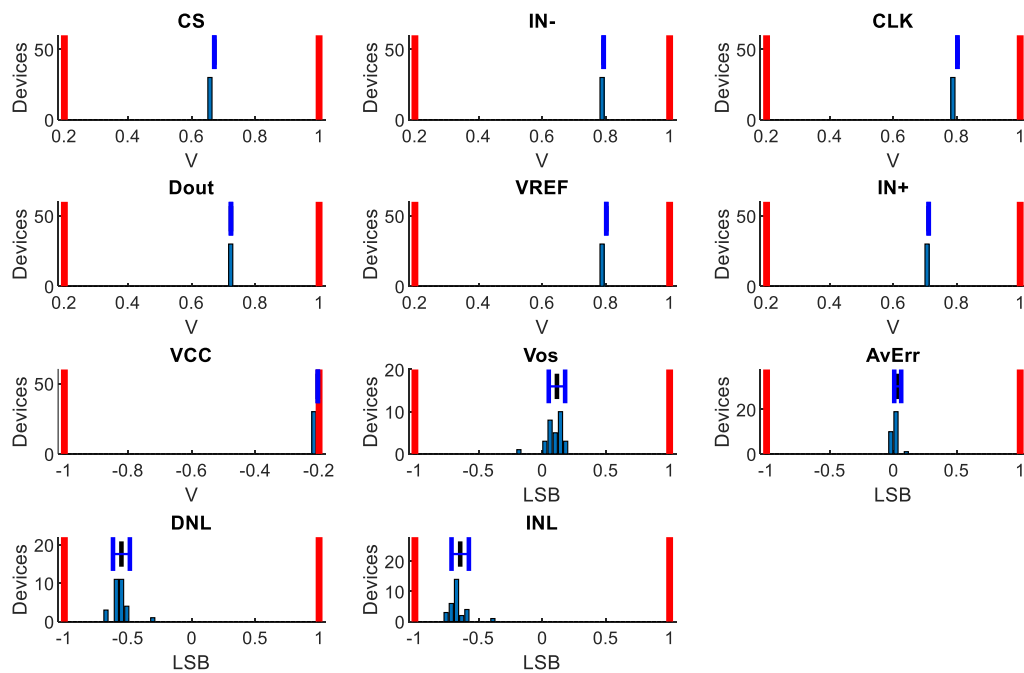


Figure 4. Histogram plots of test results for the logged data values

Analysis:

Continuity test results show fairly good repeatability, with GRR values under 1%. GRR value for Offset voltage (Vos) landed just under 10%, which is on the passing threshold. The observation is that compared to the continuity test, other tests (ZSE, FSE, DNL, and INL) generally have higher GRR values.

This result was likely due to continuity tests having more wait time for the circuit to settle once the contactors are driven. Unlike the continuity test, which involves a source-measurement process once per pin, other tests involve linear sweeps across a wide range of input values, likely requiring a loop in the code. Loops, such as while or for statements, can increase overall test time. Having a single wait statement inside these loops can add up to the testing time much faster. To compensate for this, all tests besides continuity had shorter waiting times coded.

Another assumption underlying the result is that accuracy was traded off to reduce test time for tests with multiple long loops. The linear sweep method and the histogram method both involve incrementing the input voltage values. Depending on the resolution (step size) of the input voltage, the accuracy can be affected, which could also impact repeatability. Higher accuracy could improve repeatability, but it also means a smaller step size for incrementation, so the loops in the code could take much longer. For the test time's sake, the step sizes for ZSE, FSE, DNL, and INL were 1mV, which could not have provided a high enough resolution for GRR under 10%.

User Init Code

```
msGroupSite(0, "APU%d, APU%d, SP100%d, APU%d, QMS%d", APU_VDD, APU_VIN, SPU_VIN, APU_REG, QMS_VIN);

mcuset( MCU_MAIN, 40.0 ); //40 Mhz

//digital clk set up
dpinperiod( 160 );
dpinlevel( DPU_ALL_PINS, NULL, 5, 0, 2.4, 1, MS_ALL );

dpintimeset( "CLK", "timing", "DefaultTimesetIn", DPIN_TS_PCNT, DPIN_KT, 10, 60, DPIN_IGNORE_TIMING, DPIN_IGNORE_TIMING, MS_ALL );

dpintimeset( "DOUT", "timing", "DefaultTimesetOut", DPIN_TS_PCNT, DPIN_NF, 80, DPIN_IGNORE_TIMING, DPIN_IGNORE_TIMING, DPIN_IGNORE_TIMING, MS_ALL);
dpintimeset( "CS", "timing", "DefaultTimesetIn", DPIN_TS_PCNT, DPIN_RZ, 30, 80, DPIN_IGNORE_TIMING, DPIN_IGNORE_TIMING, MS_ALL);
```

Continuity Test Code

```
ETS_PRGFLOW_FUNC Continuity( int DSIndex, LPCTSTR TestLabel )
{
    //////////////////////////////////////
    // Update test numbering information
    SetTestNumber( TestNmbr[DSIndex] );
    SetSubTestNumber( SubTestNmbr[DSIndex] );

    apuset(APU_REG, APU_FV, 5, APU_10V, APU_10MA, PIN_TO_VI); //REG 5V
    lwait(10);
    //////////////////////////////////////
    // Make connections
    // SAMPLE: hvmatonl( VinConnections );
    cbitclose(41, 42, 45, 46);
    lwait(100);
    //////////////////////////////////////
    // Apply test condition
    // SAMPLE: fssset( GP_FSS, FSS_FV, Force[DSIndex], FSS_100MA, FSS_NOCLAMP, FSS_NOCLAMP );
    apuset(APU_VDD, APU_FV, 0, APU_10V, APU_10MA, PIN_TO_VI); //gnd VCC

    dpinviset( "Continuity", DPIN_FI, 1, DPIN_8V, DPIN_8MA, DPIN_CLAMP_OFF, DPIN_CLAMP_OFF, MS_ALL );
    //////////////////////////////////////
    // Take measurements
    // SAMPLE: fssmi( GP_FSS, 50, 300 );
    RESULTS_STR continuitydata[5];
    dpinmv( "Continuity", 40, 13.0, MS_ALL );
    groupgetresults(continuitydata, 5);
    //////////////////////////////////////
    // Log the results
    for(int i = 0; i < 5; i++){
        msLogResult(0, DSIndex++, continuitydata[i].value);
    }

    apuset(APU_VDD, APU_VIOFF, 1, APU_10V, APU_10MA, PIN_TO_VI); //gnd VCC
    dpinviset( "Continuity", DPIN_OFF, 0, DPIN_8V, DPIN_8MA, DPIN_CLAMP_OFF, DPIN_CLAMP_OFF, MS_ALL );
    //dpindisconnect(MS_ALL, DPU_ALL_PINS);

    apuset(APU_VDD, APU_FV, 0, APU_10V, APU_10MA, PIN_TO_VI); //set SPU
    lwait(100);

    apuset(APU_VIN, APU_FI, 1, APU_10V, APU_10MA, PIN_TO_VI); //set APU
    lwait(10);
```

```

double vincont = apumv(APU_VIN, 4, 13); //Vin
msLogResult(0, DSIndex++, vincont);

apuset(APU_VIN, APU_FV, 0, APU_10V, APU_10MA, PIN_TO_VI); //set APU
dpinviset( "INN", DPIN_FV, 0, DPIN_8V, DPIN_8MA, DPIN_CLAMP_OFF, DPIN_CLAMP_OFF, MS_ALL );
dpinviset( "CS", DPIN_FV, 0, DPIN_8V, DPIN_8MA, DPIN_CLAMP_OFF, DPIN_CLAMP_OFF, MS_ALL );
dpinviset( "CLK", DPIN_FV, 0, DPIN_8V, DPIN_8MA, DPIN_CLAMP_OFF, DPIN_CLAMP_OFF, MS_ALL );
dpinviset( "VREF", DPIN_FV, 0, DPIN_8V, DPIN_8MA, DPIN_CLAMP_OFF, DPIN_CLAMP_OFF, MS_ALL );
lwait(100);
apuset(APU_VDD, APU_FI, -1, APU_10V, APU_10MA, PIN_TO_VI); //set APU
lwait(1000);
double vddcont = apumv(APU_VDD, 4, 13); //VDD
msLogResult(0, DSIndex++, vddcont);
//////////
// Clean Up Code
//      SAMPLE: fssset( GP_FSS, FSS_OFF, 0.0, FSS_100MA, FSS_NOCLAMP, FSS_NOCLAMP );
//      hvmatclr();

apuset(APU_VDD, APU_FV, 0, APU_10V, APU_10MA, PIN_TO_VI); //turn off APU
lwait(10);
apuset(APU_VDD, APU_VIOFF, 1, APU_10V, APU_10MA, PIN_TO_VI); //turn off APU

apuset(APU_VIN, APU_FV, 0, APU_10V, APU_10MA, PIN_TO_VI); //turn off APU
lwait(10);
apuset(APU_VIN, APU_VIOFF, 1, APU_10V, APU_10MA, PIN_TO_VI); //turn off APU

dpindisconnect(MS_ALL, DPU_ALL_PINS);

cbitopen(41, 42, 45, 46);
return( msSiteStat( MS_ALL ) ); // Return w/status
} // END_ETSPRGFLOW_FUNC

```

ZSE & FSE Code

```
ETS_PRGFLOW_FUNC VosGerr( int DSIndex, LPCTSTR TestLabel )
{
    //////////////////////////////////////
    // Update test numbering information
    SetTestNumber( TestNmbr[DSIndex] );
    SetSubTestNumber( SubTestNmbr[DSIndex] );
    apuset(APU_REG, APU_FV, 5, APU_10V, APU_10MA, PIN_TO_VI); //REG 5V
    lwait(10);
    //////////////////////////////////////
    // Make connections
    //     SAMPLE: hvmatonl( VinConnections );
    cbitclose(28, 44, 45);
    lwait(100);
    //////////////////////////////////////
    // Apply test condition
    double VLSB = 5/255.0;
    double zs_start = -VLSB; //-1 LSB
    double fs_start = 5.0 - 2.5*VLSB;
    //     SAMPLE: fssset( GP_FSS, FSS_FV, Force[DSIndex], FSS_100MA, FSS_NOCLAMP,
    apuset(APU_VDD, APU_FV, 5, APU_10V, APU_10MA, PIN_TO_VI);
    sp100set(SPU_VIN, SP_FV, zs_start, SP_10V, SP_20MA); //set SPU
    qmsset( QMS_VIN, QMS_5V, QMS_100KHZ ); //set QMS
    //qmsadcmode(QMS_VIN, QMS_ADC16);
    lwait(10);
    //mcurun(MCU_NOANALOG, "timing");
    //////////////////////////////////////
    // Take measurements
    //     SAMPLE: fssmi( GP_FSS, 50, 300 );

    double VOS;
    double FSE;
    INT64 data[8];
```

```

while(1){
    lwait(5);
    mcurun(MCU_NOANALOG, "timing");
    dpingetcapturedata("DOUT", DPIN_CAPTDATA, 8, data, 0 );
    INT64 val = 0;
    for(int i = 0; i < 8; i++){ //convert to integer
        val += (data[i] << (7-i));
    }
    if(val == 1){
        double vin = qmsmv(QMS_VIN, QMS_PREC, 4, 13.0);
        VOS = vin - (VLSB/2);
        globalVOS = VOS; //save for later
        break;
    }

    zs_start += 0.001;
    sp100set(SPU_VIN, SP_FV, zs_start, SP_10V, SP_20MA); //set SPU
}

sp100set(SPU_VIN, SP_FV, fs_start, SP_10V, SP_20MA); //set SPU
while(1){
    lwait(5);
    mcurun(MCU_NOANALOG, "timing");
    dpingetcapturedata("DOUT", DPIN_CAPTDATA, 8, data, 0 );
    INT64 val = 0;
    for(int i = 0; i < 8; i++){ //convert to integer
        val += (data[i] << (7-i));
    }
    if(val == 255){
        double vin = qmsmv(QMS_VIN, QMS_PREC, 4, 13.0);
        FSE = vin - (5.0 - VLSB*1.5);
        break;
    }

    fs_start += 0.001;
    sp100set(SPU_VIN, SP_FV, fs_start, SP_10V, SP_20MA); //set SPU
}

```

```

// Log the results
msLogResult(0, DSIndex++, VOS/VLSB);
msLogResult(0, DSIndex++, FSE/VLSB);

////////////////////////////////////
// Clean Up Code
//      SAMPLE: fssset( GP_FSS, FSS_OFF, 0.0, FSS_100MA, FSS_NOCLAMP, FSS_NOCLAMP )
//      hvmatchlr();
apuset(APU_VDD, APU_FV, 0, APU_10V, APU_10MA, PIN_TO_VI); //turn off APU
lwait(10);
apuset(APU_VDD, APU_VIOFF, 1, APU_10V, APU_10MA, PIN_TO_VI); //turn off APU

sp100set(SPU_VIN, SP_FV, 0, SP_10V, SP_20MA); //set SPU
lwait(10);
sp100set(SPU_VIN, SP_OFF, 1, SP_10V, SP_20MA); //turn off SPU0

apuset(APU_REG, APU_FV, 0, APU_10V, APU_10MA, PIN_TO_VI); //turn off APU
lwait(10);
apuset(APU_REG, APU_VIOFF, 1, APU_10V, APU_10MA, PIN_TO_VI); //turn off APU
dpindisconnect(MS_ALL, DPU_ALL_PINS);
cbitopen(28, 44, 45);
return( msSiteStat( MS_ALL ) ); // Return w/status
} // END_ETC_PRGFLOW_FUNC

```

INL & DNL Test Code

```
ETS_PRGFLOW_FUNC histogram( int DSIndex, LPCTSTR TestLabel )
{
    //////////////////////////////////////
    // Update test numbering information
    SetTestNumber( TestNmbr[DSIndex] );
    SetSubTestNumber( SubTestNmbr[DSIndex] );
    apuset(APU_REG, APU_FV, 5, APU_10V, APU_10MA, PIN_TO_VI); //REG 5V
    lwait(10);
    //////////////////////////////////////
    // Make connections
    // SAMPLE: hvmatonl( VinConnections );
    cbitclose(28, 44, 45);
    lwait(100);
    //////////////////////////////////////
    // Apply test condition
    double VLSB = 5/255.0;
    double start = globalVOS; // offset v
    double end = 5.0 - (VLSB/2) + globalVOS;
    // SAMPLE: fssset( GP_FSS, FSS_FV, Force[DSIndex], FSS_100MA, FSS_NOCLAMP, FS
    apuset(APU_VDD, APU_FV, 5, APU_10V, APU_10MA, PIN_TO_VI); //gnd VCC
    sp100set(SPU_VIN, SP_FV, start, SP_10V, SP_20MA); //set SPU
    //qmsset( QMS_VIN, QMS_5V, QMS_100KHZ ); //set QMS
    //qmsadcmode(QMS_VIN, QMS_ADC16);
    lwait(10);
    //mcurun(MCU_NOANALOG, "timing");
    //////////////////////////////////////
    // Take measurements
    // SAMPLE: fssmi( GP_FSS, 50, 300 );

    INT64 data[8];
    INT64 hist[256] = {};
    INT64 total = 0;
```

```

while(1){
    lwait(5);
    mcurun(MCU_NOANALOG, "timing");
    dpingetcapturedata("DOUT", DPIN_CAPTDATA, 8, data, 0 );
    INT64 val = 0;
    for(int i = 0; i < 8; i++){ //convert to integer
        val += (data[i] << (7-i));
    }

    hist[val] = hist[val] += 1;
    total += 1;
    if(start >= end){
        break;
    }
    start += 0.001;
    sp100set(SPU_VIN, SP_FV, start, SP_10V, SP_20MA); //set SPU
}

double avg_codewidth = total/256.0;
double current_DNL;
double max_DNL = 0;
double DNL[256];
double current_INL = 0;
double max_INL = 0;
double INL[256];
INL[0] = 0;

for (int i = 0; i < 256; i++){
    double current_codewidth = hist[i]/avg_codewidth;//codewidth
    DNL[i] = current_codewidth - 1.0;

    current_DNL = DNL[i];
    if(abs(current_DNL) > abs(max_DNL)){
        max_DNL = current_DNL;
    }

    if(i >= 1){
        INL[i] = INL[i-1]+ DNL[i-1];
        current_INL = INL[i];
        if(abs(current_INL) > abs(max_INL)){
            max_INL = current_INL;
        }
    }
}
}

```

```

// Log the results
msLogResult(0, DSIndex++, max_DNL);
msLogResult(0, DSIndex++, max_INL);

////////////////////////////////////
// Clean Up Code
//      SAMPLE: fssset( GP_FSS, FSS_OFF, 0.0, FSS_100MA, FSS_NOCLAMP, FSS_NOCLA
//                  hvmatchlr());
apuset(APU_VDD, APU_FV, 0, APU_10V, APU_10MA, PIN_TO_VI); //turn off APU
lwait(10);
apuset(APU_VDD, APU_VIOFF, 1, APU_10V, APU_10MA, PIN_TO_VI); //turn off APU

sp100set(SPU_VIN, SP_FV, 0, SP_10V, SP_20MA); //set SPU
lwait(10);
sp100set(SPU_VIN, SP_OFF, 1, SP_10V, SP_20MA); //turn off SPU0

apuset(APU_REG, APU_FV, 0, APU_10V, APU_10MA, PIN_TO_VI); //turn off APU
lwait(10);
apuset(APU_REG, APU_VIOFF, 1, APU_10V, APU_10MA, PIN_TO_VI); //turn off APU
dpindisconnect(MS_ALL, DPU_ALL_PINS);
cbitopen(28, 44, 45);
return( msSiteStat( MS_ALL ) ); // Return w/status
} // END_ETS_PRGFLOW_FUNC

```