# USING A1330 ASEK DLLS TO MAKE TWO-POINT LABVIEW PROGRAMMER

By James Zhao
Allegro MicroSystems

## INTRODUCTION

The Allegro A1330 is a 360° angle sensor IC that provides contactless high-resolution angular position based on circular vertical Hall (CVH) technology. The A1330 incorporates programmable angle scaling, angle offset, and several other registers in EEPROM to enable a full signal output to be achieved for applications requiring less than 360° rotation, such as throttle position, acceleration/brake pedal position, and other short-stroke applications. For details on how to use the A1330 in these applications, refer to the A1330 datasheet and related application notes posted on the Allegro website (https://www.allegromicro.com/en/insights-and-innovations) and download the Allegro A1330 Sample Programmer Graphical User Interface (GUI) at https://registration.allegromicro.com/#/.

The A1330 Sample Programmer GUI is written in the C# language, but when customers enter mass production (MP) phase, they commonly use LabVIEW to implement MP software. This application note provides guidelines on how to use the A1330 dynamically linked library (DLL) with LabVIEW.

Hardware and software used in this application note include:

- ASEK-20/21
- ASEK-1330-SUBKIT-T
- Operating System: Windows 10 64-bit
- LabVIEW: 2017 64-bit
- A1330 DLL: V2.4.6

Figure 1, Figure 2, and Figure 3 show typical transfer function curves of three different applications. These three curves will be implemented in this application note using the A1330 DLL with LabVIEW.

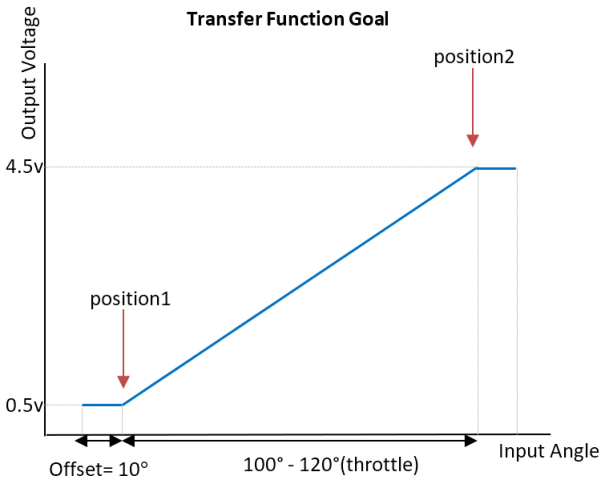


Figure 1. Single Die Throttle Application Curve

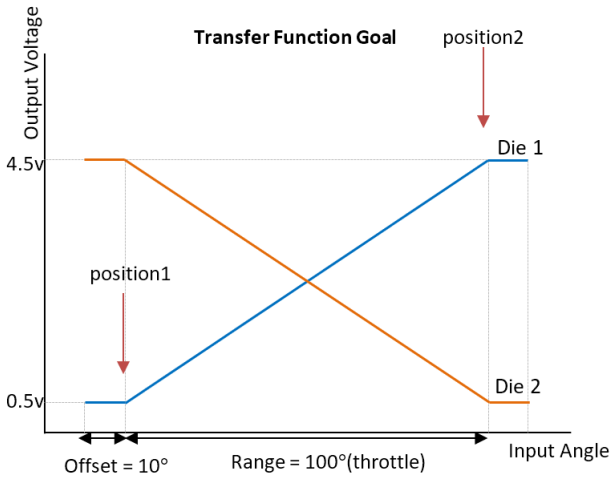


Figure 2. Dual Die Throttle Application Curve

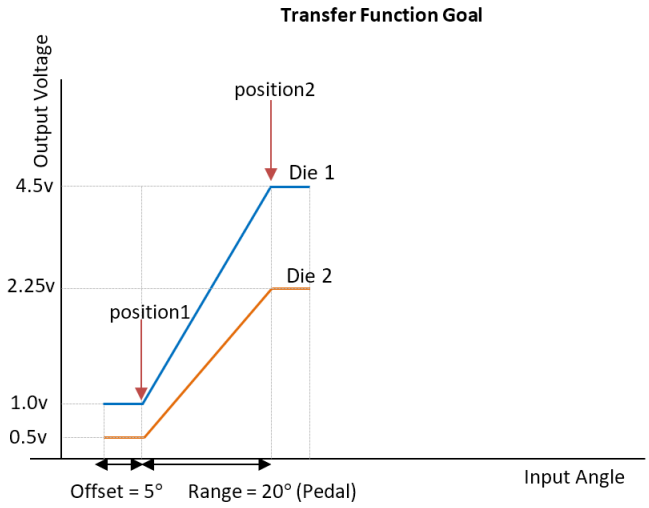


Figure 3. Dual Die Pedal Application Curve

For the above transfer functions in Figure 1, Figure 2, and Figure 3, note that if the desired low level is less than 0.35 V or the desired high level is higher than 4.65 V, some devices may not be able to reach this value because of DAC output limitations; see datasheet specification for guaranteed minimum DAC output. Also, if the desired offset multiplied by the desired gain is greater than 180°, insufficient adjustment exists in the A1330 to meet this requirement. As an example, if the desired gain is 4×, the offset cannot be larger than 45 mechanical degrees.

| Characteristics | Symbol | Test Conditions | Min. | Typ. | Max. | Units |
|---|---|---|---|---|---|---|
| Output Saturation Voltage | $V_{OUT(MAX)}$ | Max input angle position; $V_{CC}$ = 5 V, HIGH_CLAMP = 0 | 4.65 | 4.75 | – | V |
| | $V_{OUT(MIN)}$ | 0° input angle position; $V_{CC}$ = 5 V, HIGH_CLAMP = 0 | – | 0.25 | 0.35 | V |

# USING A1330 ASEK DLLS IN LABVIEW FOR TWO-POINT PROGRAMMING

## Basic Steps

Download and extract the Allegro A1330 libraries from https://registration.allegromicro.com/#/. The following files will be found in the library directory:

| Name | Date modified | Type | Size |
|---|---|---|---|
| ASEK20.dll | 8/20/2019 10:55 AM | Application extens... | 165 KB |
| ASEK20_A1330 Documentation.chm | 8/20/2019 10:55 AM | Compiled HTML ... | 578 KB |
| ASEK20_A1330.dll | 8/20/2019 10:55 AM | Application extens... | 50 KB |
| ASEKBase.dll | 8/20/2019 10:55 AM | Application extens... | 101 KB |

The ASEK20_A1330.dll is the DLL file needed for LabVIEW.

1. Open the LabVIEW window, click the ‘Connectivity’ section of ‘Functions’, then click the ‘.NET’ icon; see Figure 4.

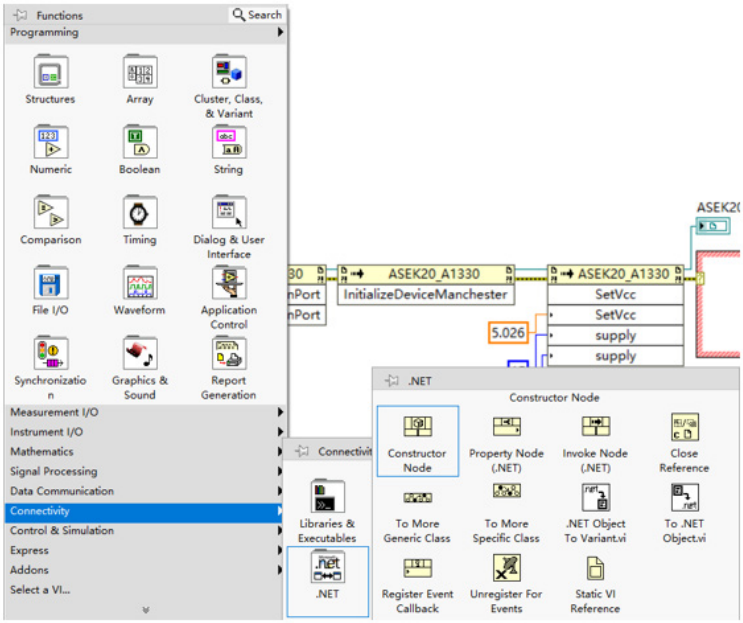


Figure 4

2. Drag the constructor node to the VI window. A “Select .NET Constructor” window will appear. Choose the Assembly Menu as ‘ASEK20_A1330’, Objects Menu as ‘ASEK20_A1330’, Constructors Menu as ‘ASEK20_A1330()’, and then click ‘OK’ button, as shown in Figure 5.

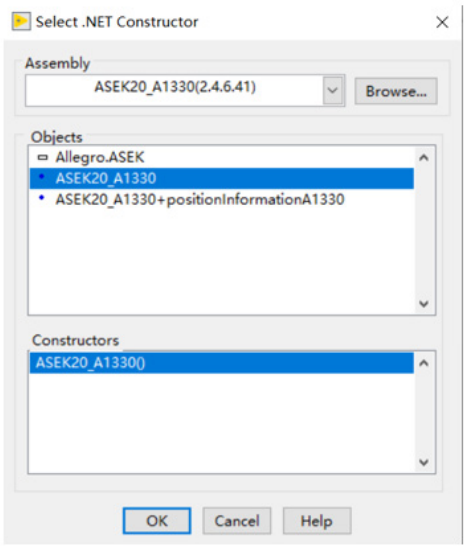


Figure 5

3. Now the constructor node should be ASEK20_A1330. In order to facilitate communication with the ASEK-20, select the communication port the ASEK-20 will use by dragging an invoke node from the .NET palette, wiring the reference from the constructor node to the reference on the invoke node, and choosing the ‘SetCommunicationPort’ as Method, as shown in Figure 6. Create a string constant with the com port name that the ASEK-20 is using and connect it to port; here, the PC port is ‘COM12’, as shown in Figure 7.

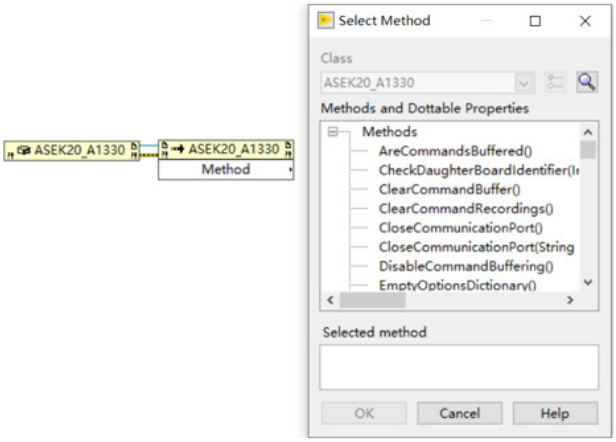


Figure 6

4. In order to communicate with the part, the communication protocol must be initialized; therefore, create another invoke node, wire it in to the reference, and select the method “InitializeDeviceManchester”, as shown in Figure 7.

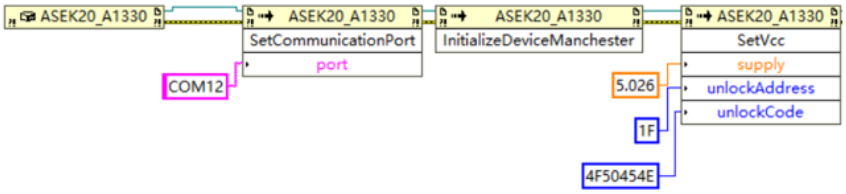


Figure 7

5. Turn on the device and create an invoke node using the ‘SetVcc’ method. This function in C# script is shown below:

```
SetVcc(double supplyVoltage, uint unlockAddress, uint unlockCode);
```

Set supplyVoltage to 5.026 to make the supply from ASEK-20/21 output 5 V. The A1330 must also have the customer code supplied during power-up to unlock the registers for reading and writing. The unlockAddress is 0x1F and unlockCode is 0x4F50454E. Create hexadecimal numeric constants with ‘0x1F’ and ‘0x4F50454E’, and connect them to corresponding inputs, as shown in Figure 7.

Now that the device is powered up, it can be written to or read from. For verification purposes, the chip ID can be read as shown in Figure 8. The ‘ReadMemory’ function in C# script is shown below:

```
ReadMemory(MemoryAccessType type, uint[] address, Progress readProgress);
```

Create an invoke node using the ‘ReadMemory’ method. There are four options that can be chosen for ‘type’: ‘primary’, ‘extended’, ‘shadow’, and ‘debug’. Choose ‘primary’ as the memory access type. Set chip ID address to ‘addr’, create a numeric array with 0x30 and 0x31, and connect them to corresponding inputs. The read-Progress is a delegate method that can be used to display a progress dialog, incrementing the progress once per memory access read. The readProgress can be null if no progress dialog is desired. The returns of this method are in an array.

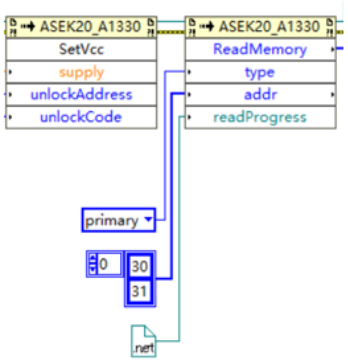


Figure 8: Read Chip ID

# IMPLEMENT SINGLE DIE TRANSFER FUNCTION

To achieve a transfer function as shown in Figure 1, use data from two readings, Position1 and Position2, to calculate the desired offset and gain values.

There are three main methods used to perform this two-point calculation and to write dedicated values to the device: TPPLGetPosition1Information, TPPLGetPosition2Information, and TPPLCalculate.

## TPPLGetPosition1Information

```
object TPPLGetPosition1Information (Dictionary<string, object> options, ProgressDone readProgress
```

This method collects the information for Position1. The following input options can be selected:

**Options** – can be null

- **Count** – integer; the number of times the inputs are sampled. Default is 1.
  - □ It is strongly recommended to sample Position1 more than once. This means it will take more time to program the A1330. Decide between desired accuracy and time to program A1330.
- **Rotation Direction** – string; when “cw”, the field is increasing clockwise and PO will be set to 0; when “ccw”, the field is increasing counterclockwise and PO will be set to 1. Default is “cw”.
  - □ This is the rotation direction of the magnet, as viewed looking down on the IC with pin 1 positioned in the top left.

**ReadProgress** – ProgressDone; a routine that updates a progress bar and can be null if not needed.

Create an invoke node and select ‘TPPLGetPosition1Information’ method. To add ‘options’, follow the procedure below:

1. Create an invoke node and select ‘EmptyOptionsDictionary’ method.
2. Create an invoke node and select ‘SetStringInOptionsDictionary’ method, wire the output of ‘EmptyOptionsDictionary’ method to ‘options’, set ‘optionName’ as ‘Rotation Direction’, and set ‘optionValue’ to ‘cw’. After these settings, the magnet can be rotated clockwise only. To rotate the magnet counterclockwise, set ‘optionValue’ to ‘ccw’.
3. Create an invoke node, select ‘SetUnsignedIntegerInOptionsDictionary’ method, set ‘optionName’ as ‘Count’, and set ‘optionValue’ to 1; this value depends on how many times to sample Position1. ‘options’ is parallel with all ‘options’ wires.
4. Connect wires like Figure 9 to get Object ‘TPPLGetPosition1Information’.

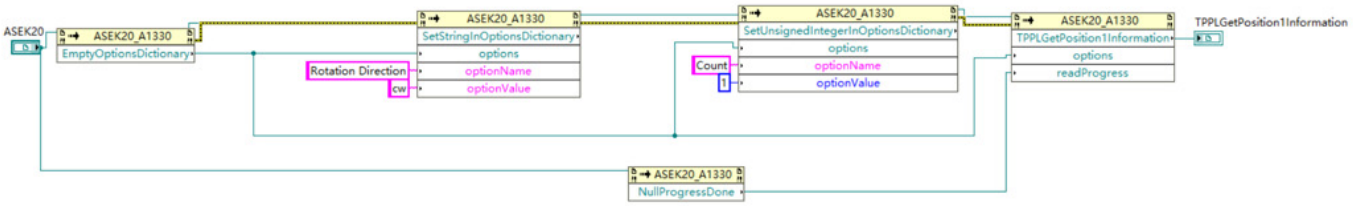


Figure 9: Get Position1 Information

## TPPLGetPosition2Information

```
object TPPLGetPosition2Information (Dictionary<string, object> options, ProgressDone readProgress)
```

This method collects the information for Position2. The following input options can be selected:

**Options – Can be Null**

- **Count** – integer; the number of times the inputs are sampled. Default is 1.
  - It is strongly recommended to sample Position2 more than once. This means it will take more time to program the A1330. Decide between desired accuracy and time to program A1330.
- **Rotation Direction** – string; when “cw”, the field is increasing clockwise and PO will be set to 0; when “ccw”, the field is increasing counterclockwise and PO will be set to 1. Default is “cw”.
  - This is the rotation direction of the magnet, as viewed looking down on the IC with pin 1 positioned in the top left.

**ReadProgress** – ProgressDone; a routine that updates a progress bar and can be null if not needed.

Create an invoke node and select ‘TPPLGetPosition2Information’ method. To add ‘options’, follow the procedure below:

1. Create an invoke node and select ‘EmptyOptionsDictionary’ method.
2. Create an invoke node and select ‘SetStringInOptionsDictionary’ method, wire the output of ‘EmptyOptionsDictionary’ method to ‘options’, set ‘optionName’ as ‘Rotation Direction’, and set ‘optionValue’ to ‘cw’. After these settings, the magnet can be rotated clockwise only. To rotate the magnet counterclockwise, set ‘optionValue’ to ‘ccw’
3. Create an invoke node, select ‘SetUnsignedIntegerInOptionsDictionary’ method, set ‘optionName’ as ‘Count’, and set ‘optionValue’ to 1; this value depends on how many times to sample Position2. ‘options’ is parallel with all ‘options’ wires.
4. Connect wires like Figure 10 to get Object ‘TPPLGetPosition2Information’.

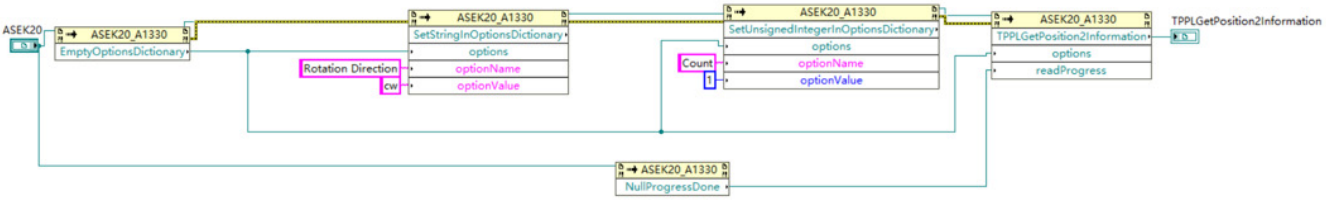


Figure 10: Get Position2 Information

## TPPLCalculate

```
Dictionary<string, uint> TPPLCalculate(object Position1, object Position2, double desiredPosition1,
  double desiredPosition2, Dictionary<string, object> options, ProgressDone calculationProgress)
```

This method takes the position information, the desired positions, and the options and calculates the values. If desired, these values are written to the device. There are six input parameters, Position1, Position2, desiredPosition1, desiresPosition2, options, and calculationProgress.

- **Position1** – the data returned from the TPPLGetPosition1Information method.
- **Position2** – the data returned from the TPPLGetPosition2Information method.
- **desiredPosition1** – the desired value for position 1. The units and value will depend on “Output Units”.
- **desiresPosition2** – the desired value for position 2. The units and value will depend on “Output Units”.
- **options** –
  - **Rotation Direction** – String; when “cw” the field is increasing clockwise and PO will be set to 0; when “ccw” the field is increasing counterclockwise and PO will be set to 1. Default is “cw”.
  - **Minimum Input Angle** – Double; the units and value will depend on “Input Units”. Default is 0.
  - **Maximum Input Angle** – Double; the units and value will depend on “Input Units”. Default is maximum value.
  - **Low Clamp Value** – Double; the units and value will depend on “Output Units”. Default is 0.
  - **High Clamp Value** – Double; the units and value will depend on “Output Units”. Default is maximum value.
  - **Post Gain Offset Value** – Double; the units and value will depend on “Input Units”. Default is 0.
  - **Clamp Enable** – Boolean; when true, the field “CE” will be set. Default is false.
  - **Rollover Enable** – Boolean; when true, the field “ROE” will be set. Default is false.
  - **Write to Chip** – Boolean; false disables writing the calculated values to the device, default is true.
  - **Input Units** – String; default is “degrees”.
    - “degrees” – all of the values are in degrees (0 to 360).
    - “percentage” – all of the values are in percentage (0 to 100).
    - “bams” – all of the values are between 0 and 4095 inclusively.
  - **Output Units** – String; default is “degrees”.
    - “degrees” – all of the values are in degrees (0 to 360).
    - “volts” – all of the values are in volts (0 to 5.0).
    - “duty cycle” – all of the values are in percentage (0 to 100).
    - “percentage” – all of the values are in percentage (0 to 100).
    - “bams” – all of the values are between 0 and 4095 inclusively.
  - **Diagnostics** – when this option is present, the internal diagnostics are enabled and returned as a string in this option.
- **calculationProgress** – ProgressDone; a routine that updates a progress bar and can be null if not needed.

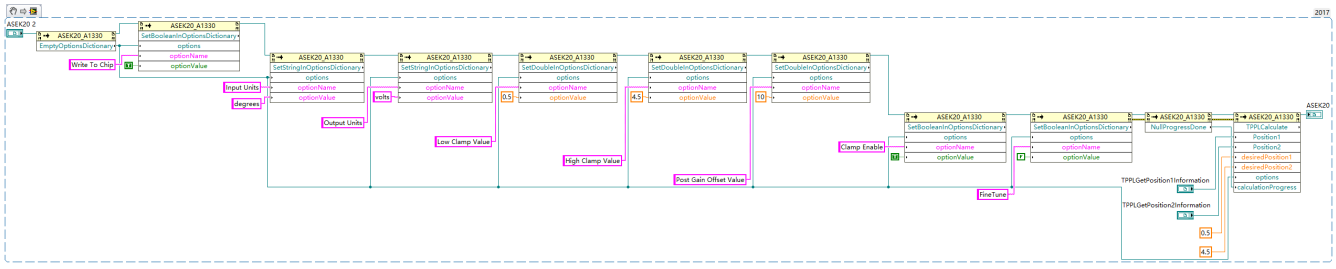

Figure 11: Calculate two-point parameters and write them to A1330.

Create an invoke node, select ‘TPPLCalculate’ method and fill arguments following below procedures:

1. Connect Object ‘TPPLGetPosition1Information’ and Object ‘TPPLGetPosition2Information’ which are implemented in Figure 9 and Figure 10 to corresponding inputs, as shown in Figure 11.
2. Create two float constants with ‘0.5’ and ‘4.5’ and wire them to ‘desiredPosition1’ and ‘desiredPosition2’ as in Figure 11.
3. Create invoke node with related method to fill the ‘options’ of TPPLCalculate(), if desired. The methods that are used depend on the type of optionValue. For example, to change the output units from default value (degrees) to voltages, select ‘SetStringInOptionsDictionary’ for this method and wire a string constant named ‘volts’, as shown in Figure 11.
4. Some options must be noted to achieve the curve in Figure 1:
   - Set ‘Low Clamp value’ to 0.5, which means the curve is clamped to 0.5 V at minimum.
   - Set ‘High Clamp value’ to 4.5, which means the curve is clamped to 4.5 V at maximum.
   - Set ‘Post Gain Offset Value’ to 10, which provides 10 degrees headroom before Position1 to stay low clamp value.
   - Set ‘Clamp Enable’ to True, enabling clamp function of A1330.
5. If desired, set ‘FineTune’ to True or False to tradeoff between accuracy and time to program A1330.

At this point, all functions have been introduced. The next step depends on how these methods will be used to make the mass production programmer. An example is given in the flowchart below:

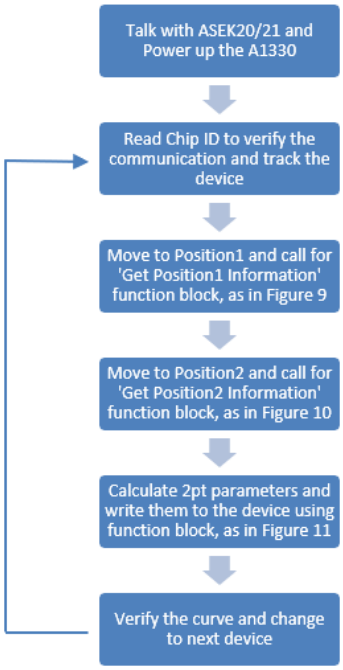


Figure 12: Two points programing Flow Chart in LabVIEW.

The flowchart above can be modified based on application requirements.

Now when moving the magnet from Position1 to Position2, the transfer function of Figure 1 should appear.

## IMPLEMENT DUAL DIE TRANSFER FUNCTION

If the A1330 dual die version is to be used to make curves like Figure 2 and Figure 3, the following procedure is suggested, based on ASEK-21:

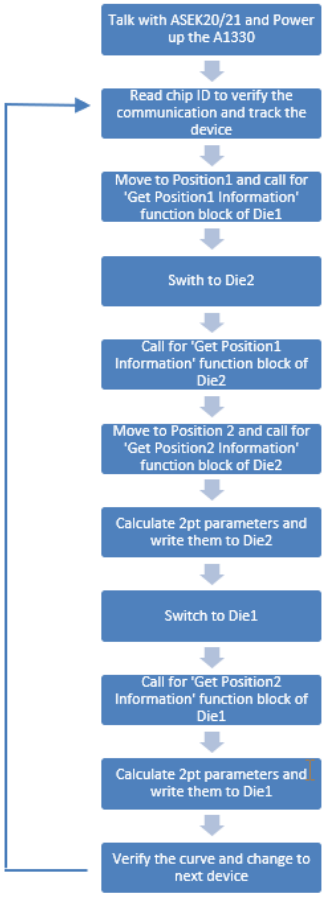


Figure 13: Implement Dual Die Transfer Function in LabVIEW.

Most functions and methods used in this flowchart have been stated previously, except how to switch dies.

On ASEK-20, use `SetDieSelection (int die)` to set which die the DLL will communicate. The input should be 0 or 1.

On ASEK-21, use `SetASEK21Port (uint port)` to set which port to use for ASEK-20 device commands. Port must be a value between 1 and 4. Refer to article about how to connect A1330 and ASEK-21 on the Allegro software website.

## CONCLUSION

Customers can perform two-point programming in LabVIEW following the procedures and flowcharts in this application note. With the Allegro Programmer ASEK-21, it is simple to support customer applications in mass production, such as throttle position, acceleration/brake pedal position, and other short-stroke (<360° rotation) applications. To learn more information about the A1330 DLL and code examples written in C# language, refer to 'Programming_the_A1330_using_the_ASEK_DLLs' article on Allegro software website.

*Revision History*

| Number | Date | Description | Responsibility |
|---|---|---|---|
| – | July 9, 2020 | Initial release | J. Zhao |



955 PERIMETER ROAD • MANCHESTER, NH 03103 • USA
+1-603-626-2300 • FAX: +1-603-641-5336 • ALLEGROMICRO.COM