|  |
| --- |
|  |
| Programming the A1330 using the ASEK DLLs |
| A guide to programming the Allegro A1330 using the ASEK DLLs |
|  |
| By K. Robert Bate |
| Allegro MicroSystems, LLC |

|  |
| --- |
|  |

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Comment** | **Responsible** |
| - | 17 June 2019 | Initial Release | K. Robert Bate |
| 1.0 | 9 July 2019 | Added die selection for ASEK-20 and Rotation Selection for TPPL | K. Robert Bate |

**Table Of Contents**

[Revision History 1](#_Toc14278841)

[1. Programming devices with the ASEK DLLs 7](#_Toc14278842)

[1.1 Hardware Requirements 7](#_Toc14278843)

[1.2 Setting Up 7](#_Toc14278844)

[1.3 Initializing 8](#_Toc14278845)

[1.3.2 Initialization for Manchester Protocol 8](#_Toc14278846)

[1.4 Setting and Getting the Power to the Device 10](#_Toc14278847)

[1.4.1 SetVccOff 10](#_Toc14278848)

[1.4.2 SetVcc(double) 10](#_Toc14278849)

[1.4.3 SetVcc(double, uint, uint) 10](#_Toc14278850)

[1.4.4 GetVcc 10](#_Toc14278851)

[1.4.5 GetIcc 10](#_Toc14278852)

[1.5 Reading and Writing Memory on the Device 10](#_Toc14278853)

[1.5.1 MemoryAccessType 10](#_Toc14278854)

[1.5.2 ReadMemory (MemoryAccessType, uint) 11](#_Toc14278855)

[1.5.3 WriteMemory(MemoryAccessType, uint, uint) 11](#_Toc14278856)

[1.5.4 ReadMemory(MemoryAccessType, uint[], Progress) 11](#_Toc14278857)

[1.5.5 WriteMemory(MemoryAccessType, uint[], uint[], Progress) 11](#_Toc14278858)

[1.6 Reading and Writing fields of the Device 11](#_Toc14278859)

[1.6.1 ReadRegister(MemoryAccessType, uint) 11](#_Toc14278860)

[1.6.2 ReadPartialRegister(MemoryAccessType, uint, int, int) 11](#_Toc14278861)

[1.6.3 ReadPartialRegisterSigned(MemoryAccessType, uint, int, int) 12](#_Toc14278862)

[1.6.4 WriteRegister(MemoryAccessType, uint, uint) 12](#_Toc14278863)

[1.6.5 WritePartialRegister(MemoryAccessType, uint, uint, int, int) 12](#_Toc14278864)

[1.7 Angle Read Operations 12](#_Toc14278865)

[1.7.1 ReadOutputAngle 12](#_Toc14278866)

[1.8 Analog Read Operations 12](#_Toc14278867)

[1.8.1 ReadOutputVoltage 12](#_Toc14278868)

[1.9 PWM Read Operations 12](#_Toc14278869)

[1.9.1 ReadOutputDutyCycle 13](#_Toc14278870)

[1.9.2 ReadOutputDutyCycle 13](#_Toc14278871)

[1.9.3 ReadOutputDutyCycleAndFrequency(out double dutyCycle, out double frequency) 13](#_Toc14278872)

[1.10 ASEK-20 Operations 13](#_Toc14278873)

[1.10.1 SetDieSelection(int) 13](#_Toc14278874)

[1.10.2 GetDieSelection() 13](#_Toc14278875)

[1.11 ASEK-20 Options 13](#_Toc14278876)

[1.11.1 UseInternalPullup 13](#_Toc14278877)

[1.11.2 UseInternalPullupVoltage 14](#_Toc14278878)

[1.11.3 UseInternalPulldown 14](#_Toc14278879)

[1.12 Two Point Programming 14](#_Toc14278880)

[1.12.1 TPPLGetPosition1Information 14](#_Toc14278881)

[1.12.2 TPPLGetPosition2Information 14](#_Toc14278882)

[1.12.3 TPPLCalculate 15](#_Toc14278883)

[1.13 ASEK-21 Operations 16](#_Toc14278884)

[1.13.1 HasASEK21 16](#_Toc14278885)

[1.13.2 SetASEK21Relay(uint, bool) 16](#_Toc14278886)

[1.13.3 SetASEK21Relays(uint, uint) 17](#_Toc14278887)

[1.13.4 SetASEK21Port(uint) 17](#_Toc14278888)

[1.14 ASEK-21 Options 17](#_Toc14278889)

[1.14.1 ASEK21BypassPowerBuffer 17](#_Toc14278890)

[1.14.2 ASEK21DSupplyVoltage 17](#_Toc14278891)

[1.14.3 ASEK21EnableManchester 17](#_Toc14278892)

[1.14.4 ASEK21PowerFeedback 17](#_Toc14278893)

[1.14.5 ASEK21Use4KPullup 18](#_Toc14278894)

[1.14.6 ASEK21UseVOutPullup 18](#_Toc14278895)

[1.14.7 ASEK21VOutIsInput 18](#_Toc14278896)

[1.14.8 ASEK21VOutPullupVoltage 18](#_Toc14278897)

[1.14.9 ASEK21UsePortVccOnly 18](#_Toc14278898)

[1.15 LabView and Other Language Support 19](#_Toc14278899)

[1.15.1 NullProgress() 19](#_Toc14278900)

[1.15.2 NullProgressDone() 19](#_Toc14278901)

[1.15.3 NullOptionsDictionary() 19](#_Toc14278902)

[1.15.4 EmptyOptionsDictionary() 19](#_Toc14278903)

[1.15.5 OptionInOptionsDictionary(Dictionary<string, object> options, string optionName) 19](#_Toc14278904)

[1.15.6 GetStringFromOptionsDictionary(Dictionary<string, object> options, string optionName) 19](#_Toc14278905)

[1.15.7 SetStringInOptionsDictionary(Dictionary<string, object> options, string optionName, string optionValue) 19](#_Toc14278906)

[1.15.8 GetDoubleFromOptionsDictionary(Dictionary<string, object> options, string optionName) 19](#_Toc14278907)

[1.15.9 SetDoubleInOptionsDictionary(Dictionary<string, object> options, string optionName, double optionValue) 20](#_Toc14278908)

[1.15.10 GetBooleanFromOptionsDictionary(Dictionary<string, object> options, string optionName) 20](#_Toc14278909)

[1.15.11 SetBooleanInOptionsDictionary(Dictionary<string, object> options, string optionName, bool optionValue) 20](#_Toc14278910)

[1.15.12 GetIntegerFromOptionsDictionary(Dictionary<string, object> options, string optionName) 20](#_Toc14278911)

[1.15.13 SetIntegerInOptionsDictionary(Dictionary<string, object> options, string optionName, int optionValue) 20](#_Toc14278912)

[1.15.14 GetUnsignedIntegerFromOptionsDictionary(Dictionary<string, object> options, string optionName) 20](#_Toc14278913)

[1.15.15 SetUnsignedIntegerInOptionsDictionary(Dictionary<string, object> options, string optionName, uint optionValue) 20](#_Toc14278914)

[1.15.16 FieldInTPPLResultsDictionary(Dictionary<string, uint> results, string fieldName) 20](#_Toc14278915)

[1.15.17 FieldsInTPPLResultsDictionary(Dictionary<string, uint> results) 20](#_Toc14278916)

[1.15.18 GetUnsignedIntegerFromTPPLResultsDictionary(Dictionary<string, uint> results, string fieldName) 21](#_Toc14278917)

[1.16 Optimizing 21](#_Toc14278918)

[1.16.1 Use the Array variant of ReadMemory and WriteMemory 21](#_Toc14278919)

[1.16.2 Do not use WritePartialRegister 21](#_Toc14278920)

[1.16.3 Use EnableCommandBuffering 21](#_Toc14278921)

[2. Examples 21](#_Toc14278922)

[2.1 Example for A1330 in Analog mode 21](#_Toc14278923)

[2.2 Example for A1330 in PWM mode 23](#_Toc14278924)

[2.3 Two Point Programming Example, Input in voltage 25](#_Toc14278925)

# Programming devices with the ASEK DLLs

## Hardware Requirements

To program the A1330 using the ASEK DLLs you need one of the following:

An ASEK-1330-T-KIT which contains:

* ASEK-20-T-KIT
  + ASEK-20 Chassis with main Motherboard inside (85-0540-004)
  + USB Communications Cable
  + DC Power Supply/Cable with AC Outlet Adapters
  + Proto Board (Part # 85-0540-103) (Not required with A1330 communication)
  + Ribbon Cable (Part # 85-0540-300)
* ASEK-1330-SUBKIT-T
  + A1330 Daughterboard (Part #: TED-0002219 / Stenciled A1330)
  + A1330 Socketed grand-daughter board (Part #: 85-0808-001 / Stenciled ASEK1330-SMT)
  + A1330 Surface mount grand-daughter board (Part #: 85-0808-004)

Or

* ASEK-1330-SUBKIT-T
  + A1330 Daughterboard (Part #: TED-0002582 / Stenciled A1330)
  + A1330 Socketed grand-daughter board (Part #: 85-0808-001 / Stenciled ASEK1330-SMT)
  + A1330 Surface mount grand-daughter board (Part #: 85-0808-004)

Or

An ASEK21-T-KIT which contains:

* ASEK-21 Main Board(Part #: 85-0712)
* ASEK-21 Calibration Board(Part #: 85-0712-100)
* ASEK-21 Daughter Board(Part #: 85-0712-103)
* ASEK-21 Cable(Part #: 85-0712-301)
* USB A male to B male 3-foot cable (Part#: 3021001-03)
* ASEK-21 Power Supply (Part #: EMMA050400-P5P-IC)

## Setting Up

If you are using Visual Studio to build the application, the first step is to add references to the DLLs. The ASEK-21 is an ASEK-20 with some added capability and is treated as an ASEK-20.

All the DLLs will require a reference to ASEKBase.dll. Right click on the project icon in the Solution Explorer; select “Add Reference…” from the menu. In the Add reference dialog, use the Browse tab to navigate to the ASEKBase.dll and selected it then click on the OK button. Perform the same actions to add ASEK20.dll and ASEK20\_A1330.dll.

## Initializing

To initialize the programmer, create an object of the type you want to communicate with. The class that you will be instantiating will be composed of two parts. The first is the name of the ASEK that will be used for communication and the second is the name of the device. In this case you would write:

ASEK20\_A1330 device = new ASEK20\_A1330();

Next, you must tell the object which of the ASEKs that are connected to your computer you want to use. This will depend on which ASEK you are using. For the ASEK-20 you need to tell it which COM port is operating.

device.SetCommunicationPort(string portName); // for ASEK-20

Once you have created the object and set which port or address is to be used, you need to initialize it to utilize the communication protocol desired. The A1330 has one protocol, Manchester.

device.InitializeDeviceManchester();

For each protocol there will be a set of methods to change the defaults for that protocol and they and the values of the parameters they need are device dependent and are covered in the help file for the device DLL.

#### IsAnalogOutput

Set this to true if the A1330 is setup to have an analog output. Default is true.

device.IsAnalogOutput = true;

### Initialization for Manchester Protocol

#### SetManchesterHighVoltage

The A1330 receives the Manchester commands on the Vcc line. This method sets the high voltage used during that communication in volts. The default value is 8 volts.

device.SetManchesterHighVoltage(double voltage);

#### GetManchesterHighVoltage

The A1330 receives the Manchester commands on the Vcc line. This method gets the high voltage used during that communication in volts. The default value is 8 volts.

Double voltage = device.GetManchesterHighVoltage();

#### SetManchesterLowVoltage

The A1330 receives the Manchester commands on the Vcc line. This method sets the low voltage used during that communication in volts. The default value is 6 volts.

device.SetManchesterLowVoltage(double voltage);

#### GetManchesterLowVoltage

The A1330 receives the Manchester commands on the Vcc line. This method gets the low voltage used during that communication in volts. The default value is 6 volts.

Double voltage = device.GetManchesterLowVoltage();

#### SetManchesterSlewRate(double)

This method sets the slew rate when communication with the device. The units of rate are volts per microsecond and must be within the limits specified by the device. Note: If the communication speed is set to be greater than 40000 then the slew rate controls are disabled. The default value is 0.75 V/uS.

device.SetManchesterSlewRate(double rate);

#### GetManchesterSlewRate()

This method returns the slew rate on communication with the device.

double rate = device.GetManchesterSlewRate();

#### SetManchesterCommunicationSpeed(double)

This method sets the speed on communication with the device. The units of “speed” are in bits per second and must be within the limits specified by the device. Note: If the speed is set to be greater than 40000 then the slew rate controls are disabled.

device.SetManchesterCommunicationSpeed(double speed);

#### GetManchesterCommunicationSpeed ()

This method returns the speed on communication with the device.

double speed = device.GetManchesterCommunicationSpeed();

#### SetManchesterInputSamplingThreshold(double)

This method sets the voltage at which the Manchester input circuitry assigns the input as a 1 or 0. The input is in volts.

device.SetManchesterInputSamplingThreshold(double voltage);

#### GetManchesterInputSamplingThreshold()

This method gets the voltage at which the Manchester input circuitry assigns the input as a 1 or 0. The output is in volts.

double voltage = device.GetManchesterInputSamplingThreshold();

## Setting and Getting the Power to the Device

There are three methods that set the voltage.

### SetVccOff

This method sets the voltage being supplied to the device to 0 volts.

device.SetVccOff();

### SetVcc(double)

This method sets the voltage being supplied to the device to what is specified by the parameter “supplyVoltage”. The units of “supplyVoltage” are in volts. It does not send the access codes so the registers are not accessible

device.SetVcc(double supplyVoltage);

### SetVcc(double, uint, uint)

This device needs to have the customer code supplied during power up to unlock the registers for reading and writing so this method will perform that operation. The unlock address is 0x1F and unlock code is 0x4F50454E.

device.SetVcc(double supplyVoltage, uint unlockAddress, uint unlockCode);

### GetVcc

This method gets the voltage being supplied to the device in volts.

double voltage = device.GetVcc();

### GetIcc

This method gets the current being supplied to the device in milliamps. This is an uncalibrated reading and should be used as a qualitative indicator.

double current = device.SetIcc();

## Reading and Writing Memory on the Device

The memory structure for the A1330 is composed entirely of the primary access type when reading or writing memory.

### MemoryAccessType

The type MemoryAccessType is used to specify which type of access to use. The values that should be used are MemoryAccessType.primary or MemoryAccessType.shadow.

There are four commands to read and write the memory on the device.

### ReadMemory (MemoryAccessType, uint)

Reads the contents of the memory address specified.

uint content = device.ReadMemory(MemoryAccessType type, uint address);

### WriteMemory(MemoryAccessType, uint, uint)

Replaces the contents of the address specified with the data supplied.

device.WriteMemory(MemoryAccessType type, uint address, uint data);

There is some overhead when accessing the device. To decrease the time used, versions of the read and write methods were created which access a bunch of memory locations and only must perform some of the overhead once rather than once per access.

### ReadMemory(MemoryAccessType, uint[], Progress)

Uint[] contents = device.ReadMemory(MemoryAccessType type, uint[] address, Progress readProgress);

Reads the contents of the memory addresses specified and returns the contents in an array. The readProgress is a delegate method which 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.

### WriteMemory(MemoryAccessType, uint[], uint[], Progress)

device.WriteMemory(MemoryAccessType type, uint[] address, uint[] data, Progress writeProgress);

Writes the contents of the data array to the memory addresses specified. The writeProgress is a delegate method which can be used to display a progress dialog, incrementing the progress once per memory access written. The writeProgress can be null if no progress dialog is desired.

## Reading and Writing fields of the Device

To make programming the device easier, there are several methods which will operate on a portion of the memory location. These methods can be the slowest way to access memory so if speed is required, use the read and write memory methods that use arrays.

### ReadRegister(MemoryAccessType, uint)

Reads a memory location from the device, this command is the same as ReadMemory.

uint data = device.ReadRegister(MemoryAccessType type, uint address);

### ReadPartialRegister(MemoryAccessType, uint, int, int)

Reads a memory location from the device then extracts the desired bit field and right justifies it. The first two parameters are the same as ReadRegister while bits\_hi is the highest bit number in the bit field and bits\_lo is the lowest bit number in the bit field.

uint data = device.ReadPartialRegister(MemoryAccessType type, uint address, int bits\_hi, int bits\_lo);

### ReadPartialRegisterSigned(MemoryAccessType, uint, int, int)

Reads a memory location from the device then extracts the desired bit field, right justifies it then sign extends it. The first two parameters are the same as ReadRegister while bits\_hi is the highest bit number in the bit field and bits\_lo is the lowest bit number in the bit field.

int data = device.ReadPartialRegisterSigned(MemoryAccessType type, uint address, int bits\_hi, int bits\_lo);

### WriteRegister(MemoryAccessType, uint, uint)

Writes the data to the memory location on the device, this command is the same as WriteMemory.

device.WriteRegister(MemoryAccessType type, uint address, uint data);

### WritePartialRegister(MemoryAccessType, uint, uint, int, int)

Reads a memory location from the device then inserts the desired bit field data into it and then writes the contents back into the memory location. The first two parameters are the same as WriteRegister while bits\_hi is the highest bit number in the bit field and bits\_lo is the lowest bit number in the bit field. The input data is right justified.

device.WritePartialRegister(MemoryAccessType type, uint address, uint data, int bits\_hi, int bits\_lo);

## Angle Read Operations

The A1330 has one read method for reading the angle output:

### ReadOutputAngle

Reading the angle output of the device is accomplished as follows.

double value = device.ReadOutputAngle();

## Analog Read Operations

The A1330 has one read method for reading voltage output:

### ReadOutputVoltage

Reading the voltage output of the device is accomplished as follows.

double value = device.ReadOutputVoltage();

## PWM Read Operations

The A1330 has 3 read methods for reading PWM output:

### ReadOutputDutyCycle

Reading the duty cycle output of the device is accomplished as follows.

double value = device.ReadOutputDutyCycle();

### ReadOutputDutyCycle

Reading the frequency output of the device is accomplished as follows.

double value = device.ReadOutputFrequency();

### ReadOutputDutyCycleAndFrequency(out double dutyCycle, out double frequency)

Reading the duty cycle and frequency output of the device is accomplished as follows.

double dutyCycle;

double frequency;

device.ReadOutputDutyCycle(out dutyCycle, out frequency);

## ASEK-20 Operations

When using the ASEK-20 with the ASEK1330 (TED-0002219) daughterboard, there are two methods that are used for die selection.

### SetDieSelection(int)

This method sets which die the DLL is going to talk to. The input should be 0 or 1.

device.SetDieSelection(int die);

### GetDieSelection()

This method gets which die the DLL is currently talking to. The return value will be 0 or 1.

int die = device.GetDieSelection();

## ASEK-20 Options

When using the ASEK-20 with the ASEK1330 (TED-0002219) daughterboard, there are three parameters that can be set.

### UseInternalPullup

This parameter chooses if a pullup resister will be used on the Vout.

* false – No internal pullup
* true – Internal 4.75k Ω pullup resister

device.UseInternalPullup = true;

### UseInternalPullupVoltage

This parameter sets the voltage used if UseInternalPullup is true..

device.UseInternalPullupVoltage = 5.0;

### UseInternalPulldown

This parameter chooses if a pulldown resister will be used on the Vout.

* false – No internal pulldown
* true – Internal 10.0k Ω pulldown resister

device.UseInternalPulldown = true;

## Two Point Programming

Two point programming uses the data from two readings to cal­culate the desired offset and gain values.

### TPPLGetPosition1Information

This method collects the information for position 1.

#### Inputs

* options – can be null
  + Count – int, the number of times the inputs are sampled. Default is 1.
  + Rotation Direction – String, when “cw” the field is increasing clockwise and cordic\_pol will be set to 0, when “ccw” the field is increasing counter clockwise and cordic\_pol will be set to 1. Default is “cw”.
* readProgress – ProgressDone, is a routine that updates a progress bar and can be null if not wanted.

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

### TPPLGetPosition2Information

This method collects the information for position 2.

#### Inputs

* options – can be null
  + Count – int, the number of times the inputs are sampled. Default is 1.
  + Rotation Direction – String, when “cw” the field is increasing clockwise and cordic\_pol will be set to 0, when “ccw” the field is increasing counter clockwise and cordic\_pol will be set to 1. Default is “cw”.
* readProgress – ProgressDone, is a routine that updates a progress bar and can be null if not wanted.

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

### TPPLCalculate

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.

#### Inputs

* Position1 – the data returned from the TPPLGetPosition1Information method.
* Position2 – the data returned from the TPPLGetPosition2Information method.
* desiredPosition1 – the desired value for position 1. Default units are % Duty Cycle.
* desiresPosition2 – the desired value for position 2. Default units are % Duty Cycle.
* options -
  + Rotation Direction – String, when “cw” the field is increasing clockwise and cordic\_pol will be set to 0, when “ccw” the field is increasing counter clockwise and cordic\_pol 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, is a routine that updates a progress bar and can be null if not wanted.

#### Outputs

* PREGAIN\_OFFSET@EEPROM
* SS@EEPROM
* GAIN@EEPROM
* CE@EEPROM
* ROE@EEPROM
* PO@EEPROM
* POSTGAIN\_OFFSET@EEPROM
* HIGH\_CLAMP@EEPROM
* LOW\_CLAMP@EEPROM
* MAX\_INPUT@EEPROM
* MIN\_INPUT@EEPROM

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

## ASEK-21 Operations

The ASEK-21 is an expansion daughterboard for the ASEK-20 that increases the power sourcing capability and multiplexes four ports into the ASEK-20’s one port.

### HasASEK21

This method returns true if the ASEK-20 has an ASEK-21 attached.

bool hasAsek21 = device.HasASEK21();

### SetASEK21Relay(uint, bool)

This method sets the relay on (true) or off (false) based on enabled. Relay is a number between 0 and 79.

device.SetASEK21Relay(uint relay, bool enabled);

### SetASEK21Relays(uint, uint)

This method sets a block of relays on or off based on the value of relayFlags. RelayBlock must be number between 0 and 4. To determine which RelayBlock a relay is in just divide the relay number by 16.

device.SetASEK21Relays(uint relayBlock, uint relayFlags);

### SetASEK21Port(uint)

This method sets which port is being used for the ASEK-20 device commands. Port must be a value between 1 and 4.

device.SetASEK21Port(uint port);

## ASEK-21 Options

### ASEK21BypassPowerBuffer

When true, the device power is directly supplied by the ASEK-20.

Warning: there is a 40ma limit to the ASEK-20 current sinking capability.

device.ASEK21BypassPowerBuffer = false;

bool bypassBufferState = device.ASEK21BypassPowerBuffer;

### ASEK21DSupplyVoltage

Sets the voltage supplied by the DSupply pin:

* 0 - None
* 1 - 5 Volts
* 2 - 3.3 Volts
* 3 - 2.7 Volts

device.ASEK21DSupplyVoltage = 2; // 3.3 volts

int dsupplyVoltage = device.ASEK21DSupplyVoltage;

### ASEK21EnableManchester

This parameter enables the use of the Manchester communication protocol. Typically it will be set by the InitializeDeviceManchester routine of the device.

device.ASEK21EnableManchester = true; // enable Manchester support

bool enableManchester = device.ASEK21EnableManchester;

### ASEK21PowerFeedback

When using the power buffer, sets which feedback loop to use:

* 0 - Most stable loop, loss of 10mV per mA required by the devices under test.
* 1 - Stable loop internal to ASEK-21. (Recommended)
* 2 - Uses the Vcc\_S line from the devices under test. Most accurate.

device.ASEK21PowerFeedback = 1; // Stable loop

int feedback = device.ASEK21PowerFeedback;

### ASEK21Use4KPullup

When set to true and ASEK21UseVOutPullup is true then the pull-up resister used is the 4.75K. If ASEK21UseVOutPullup is true and this parameter is not true then the 1K resister is used.

device.ASEK21Use4KPullup = true; // enable the 4K pullup

bool using4KPullup = device.ASEK21Use4KPullup;

### ASEK21UseVOutPullup

When set to true, one of the two pull-up resisters is connected to the VOut line. If ASEK21Use4KPullup is not set then the 1K pull-up resister is used.

device.ASEK21UseVOutPullup = true; // enable the VOut pullup

bool usingVOutPullup = device.ASEK21UseVOutPullup;

### ASEK21VOutIsInput

When set to true, the VOut line is setup as an input.

device.ASEK21VOutIsInput = true; // enable VOut input

bool voutInput = device.ASEK21VOutIsInput;

### ASEK21VOutPullupVoltage

This parameter if the voltage that will be used if VOut is pulled up. Depends on ASEK21UseVOutPullup getting set to true.

device.ASEK21VOutPullupVoltage = 5.0; // pullup VOut to 5.0 volts

double voutPullupVoltage = device.ASEK21VOutPullupVoltage;

### ASEK21UsePortVccOnly

This parameter is used to force the Vcc and Gnd lines to be from the designated port. SetASEK21Port will then only change the VOut lines.

* 0 – Always use the power and ground lines specified by SetASEK21Port.
* 1 – Always use the power and ground lines on port 1
* 2 – Always use the power and ground lines on port 2
* 3 – Always use the power and ground lines on port 3
* 4 – Always use the power and ground lines on port 4

device.ASEK21UsePortVccOnly = 3; // use Vcc and Gnd from port 3

uint usePortVccOnly = device.ASEK21UsePortVccOnly;

## LabView and Other Language Support

### NullProgress()

Returns a null cast as a Progress delegate. Useful when needing to pass a progress routine into any of the routines which accept a null argument but the language or environment currently being used does not have the capability to pass in null.

### NullProgressDone()

Returns a null cast as a ProgressDone delegate. Useful when needing to pass a progress done routine into any of the routines which accept a null argument but the language or environment currently being used does not have the capability to pass in null.

### NullOptionsDictionary()

Returns a null cast as a Dictionary<string.object>. Useful when needing to pass an options dictionary into one of the two point programming routines which accept a null argument but the language or environment currently being used does not have the capability to pass in null.

### EmptyOptionsDictionary()

Returns an empty Dictionary<string,object>. Useful when needing to pass an options dictionary into one of the two point programming routines and the language or environment currently being used does not support generic dictionary creation.

### OptionInOptionsDictionary(Dictionary<string, object> options, string optionName)

Checks the dictionary passed in for the option named. If the dictionary is null or does not contain the option returns false otherwise returns true.

### GetStringFromOptionsDictionary(Dictionary<string, object> options, string optionName)

Checks the dictionary passed in for the option named. If the dictionary is null or does not contain the option returns an empty string otherwise returns the contents of the string.

### SetStringInOptionsDictionary(Dictionary<string, object> options, string optionName, string optionValue)

Set the option to the given value in the dictionary.

### GetDoubleFromOptionsDictionary(Dictionary<string, object> options, string optionName)

Checks the dictionary passed in for the option named. If the dictionary is null or does not contain the option returns a 0.0 otherwise returns the value of the option.

### SetDoubleInOptionsDictionary(Dictionary<string, object> options, string optionName, double optionValue)

Set the option to the given value in the dictionary.

### GetBooleanFromOptionsDictionary(Dictionary<string, object> options, string optionName)

Checks the dictionary passed in for the option named. If the dictionary is null or does not contain the option returns a false otherwise returns the value of the option.

### SetBooleanInOptionsDictionary(Dictionary<string, object> options, string optionName, bool optionValue)

Set the option to the given value in the dictionary.

### GetIntegerFromOptionsDictionary(Dictionary<string, object> options, string optionName)

Checks the dictionary passed in for the option named. If the dictionary is null or does not contain the option returns a 0 otherwise returns the value of the option.

### SetIntegerInOptionsDictionary(Dictionary<string, object> options, string optionName, int optionValue)

Set the option to the given value in the dictionary.

### GetUnsignedIntegerFromOptionsDictionary(Dictionary<string, object> options, string optionName)

Checks the dictionary passed in for the option named. If the dictionary is null or does not contain the option returns a 0 otherwise returns the value of the option.

### SetUnsignedIntegerInOptionsDictionary(Dictionary<string, object> options, string optionName, uint optionValue)

Set the option to the given value in the dictionary.

### FieldInTPPLResultsDictionary(Dictionary<string, uint> results, string fieldName)

Checks the results dictionary passed in for the field named. If the dictionary is null or does not contain the field returns false otherwise returns true.

### FieldsInTPPLResultsDictionary(Dictionary<string, uint> results)

Returns an array of all of the field names in the results dictionary. If the results dictionary is null or empty, returns a zero length array.

### GetUnsignedIntegerFromTPPLResultsDictionary(Dictionary<string, uint> results, string fieldName)

Checks the results dictionary passed in for the field named. If the dictionary is null or does not contain the field returns a 0 otherwise returns the value of the field.

## Optimizing

Use these suggestions to speed up the programming of the devices.

### Use the Array variant of ReadMemory and WriteMemory

There is some overhead in setting up to read or write the registers and when the array versions of these commands are utilized, this overhead can be done at the start of the read or write command and not for every register.

### Do not use WritePartialRegister

WritePartialRegister is a method which performs a read-modify-write operation. If you are modifying a number of parameters that are in the same register, it is better to read the register into a local copy, modify the parameters in the local copy and then write the local copy back to the device.

### Use EnableCommandBuffering

When using the ASEK-20/ASEK-21, commands can be packed into a command buffer to reduce the total time from when the first command is operated on to when the last command is complete. It takes about 100mS from the time a command is sent to the ASEK-20/ASEK-21 until a response is returned. Very often, the ASEK-20/ASEK21 can perform a full command buffer’s worth of commands in less than 100mS so the total time is still 100mS. When command buffering is on, the call FlushCommandBuffer is used to force the command buffer to be sent to the ASEK-20/ASEK-21. For any read command, the command buffer is automatically flushed.

# Examples

## Example for A1330 in Analog mode

The hardware for this example is the ASEK-20 connected to the ASEK-1330 board. This example will initialize Manchester communication with an A1330 which is in analog mode. Once communication is established the output will be read. In EEPROM the gain field will be read and written. And finally, the power is turned off.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Allegro.ASEK;

namespace ASEK20\_VCCExample

{

class Program

{

static void Main(string[] args)

{

ASEK20\_A1330 device = new ASEK20\_A1330();

// Initialize the ASEK20

try

{

device.SetCommunicationPort("COM4");

device.InitializeDeviceManchester();

device.SetManchesterHighVoltage(8.0);

device.SetManchesterLowVoltage(6.0);

device.SetManchesterSlewRate(1.0);

device.SetManchesterCommunicationSpeed(40000);

device.UseInternalPullup = false;

device.UseInternalPulldown = false;

device.IsAnalogOutput = true;

device.SetManchesterInputSamplingThreshold(2.5);

device.SetVcc(5.0, 0x1F, 0x4F50454E);

}

catch (Exception ex)

{

// there was an error with the command

Console.WriteLine("Unable to Initialize the Device. Message = " + ex.Message);

return;

}

try

{

// Read the output voltage of the device.

double outputVoltage = device.ReadOutputVoltage();

Console.WriteLine("The output voltage = " + outputVoltage.ToString());

// Read the angle gain field from the device.

uint angle\_gain = device.ReadMemory(MemoryAccessType.primary, 0x3B);

Console.WriteLine("angle\_gain = " + (angle\_gain & 0x01FFF).ToString());

// Set the angle\_gain field to 1.0.

angle\_gain = (angle\_gain & 0x3FFE000) | 0;

// Write the angle\_gain field in the EEPROM on the device.

device.WriteMemory(MemoryAccessType.primary, 0x3B, angle\_gain);

// Turn the device off

device.SetVccOff();

}

catch (Exception ex)

{

// there was an error with one of the commands

Console.WriteLine("Error. message = " + ex.Message);

}

}

}

}

## Example for A1330 in PWM mode

The hardware for this example is the ASEK-21 connected on port 1 to a module that contains the A1330. See the user manual for the correct wiring between the A1330 and the ASEK-21 This example will initialize Manchester communication with an A1330. Once communication is established the output and the PWM output will be read. In EEPROM the customer field will be read and written. And finally, the power is turned off.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Allegro.ASEK;

namespace ASEK20\_SENTExample

{

class Program

{

static void Main(string[] args)

{

ASEK20\_A1330 device = new ASEK20\_A1330();

// Initialize the ASEK20

try

{

device.SetCommunicationPort("COM3");

device.InitializeDeviceManchester();

device.SetManchesterHighVoltage(8.0);

device.SetManchesterLowVoltage(6.0);

device.SetManchesterSlewRate(1.0);

device.SetManchesterCommunicationSpeed(40000);

device.IsAnalogOutput = false;

device.SetManchesterInputSamplingThreshold(2.5);

device.SetPWMInputSamplingThreshold(2.5);

device.SetVcc(5.0, 0x1F, 0x4F50454E);

device.ASEK21BypassPowerBuffer = false;

device.ASEK21PowerFeedback = 1;

device.ASEK21DSupplyVoltage = 2;

device.ASEK21UsePortVccOnly = 0;

device.ASEK21UseVOutPullup = true;

device.ASEK21VoutPullupVoltage = 5.0;

device.ASEK21Use4KPullup = false;

device.SetASEK21Port(1);

device.SetVcc(5.0, 0x86, 0xC2D8E67A);

}

catch (Exception ex)

{

// there was an error with the command

Console.WriteLine("Unable to Initialize the Device. Message = " + ex.Message);

return;

}

try

{

// Read the output duty cycle of the device.

double outputDutyCycle = device.ReadOutputDutyCycle();

Console.WriteLine("The output duty cycle = " + outputDutyCycle.ToString());

// Read the customer field from the device.

uint customer = device.ReadMemory(MemoryAccessType.primary, 0x3F);

Console.WriteLine("customer = " + customer.ToString());

// Set the customer field to 0x1234.

customer = 0x1234;

// Write the customer field in the EEPROM on the device.

device.WriteMemory(MemoryAccessType.primary, 0x3F, customer);

// Turn the device off

device.SetVccOff();

}

catch (Exception ex)

{

// there was an error with one of the commands

Console.WriteLine("Error. message = " + ex.Message);

}

}

}

}

## Two Point **Programming** Example, Input in voltage

This example will read the two positions. From those and the desired positions in voltage, calculate the new values and write them to the device.

Dictionary<string, object> options = new Dictionary<string, object>();

double desiredPosition1 = 0.5;

double desiredPosition2 = 4.5;

// Move the target to position 1

object position1Data = device.TPPLGetPosition1Information(null, null);

// Move the target to position 2

object position2Data = device.TPPLGetPosition2Information(null, null);

options["Write To Chip"] = true;

options["Low Clamp Value"] = 0.4;

options["High Clamp Value"] = 4.6;

options["Post Gain Offset Value"] = 90.0;

options["Clamp Enable"] = true;

options["Rollover Enable"] = false;

options["Input Units"] = "degrees";

options["Output Units"] = "volts";

options["Diagnostics"] = true;

fieldsChanged = device.TPPLCalculate(position1Data, position2Data, desiredPosition1, desiredPosition2, options, null);

// If the values were written to the device then this is not needed

if (fieldsChanged != null)

{

foreach (KeyValuePair<string, uint> fieldChanged in fieldsChanged)

{

string[] fieldNameAndGroup = fieldChanged.Key.Split('@');

string groupName = fieldNameAndGroup[1];

string fieldName = fieldNameAndGroup[0];

uint fieldValue = fieldChanged.Value;

// Write the values to the device

}

}

object diagnosticData = null;

if (options.TryGetValue("Diagnostics", out diagnosticData))

{

// Save the info to a file

SaveFileDialog saveMemoryFileDialog = new SaveFileDialog();

saveMemoryFileDialog.Filter = "Text Files (\*.txt)|\*.txt|All Files (\*.\*)|\*.\*";

saveMemoryFileDialog.FilterIndex = 1;

saveMemoryFileDialog.Title = "Save diagnostic information to a file";

saveMemoryFileDialog.FileName = "A1330 Two Point Programming Diagnostics";

if (saveMemoryFileDialog.ShowDialog() == DialogResult.OK)

{

File.WriteAllText(saveMemoryFileDialog.FileName, diagnosticData.ToString());

}

}