Texas Instruments, Inc.
C2000 Systems and Applications

# Digital Motor Control – Resolver Interface

## Software Library

2014

# Contents

DIGITAL MOTOR CONTROL

# Resolver Interface Software Library

## Introduction

The digital motor control library is composed of C functions (or macros) developed for C2000 motor control users. These modules are represented as modular blocks in C2000 literature in order to explain system-level block diagrams clearly by means of software modularity. The DMC library modules cover nearly all of the target-independent mathematical macros and target-specific peripheral configuration macros, which are essential for motor control. These modules can be classified as:

| | |
|---|---|
| **Transformation and Observer Modules** | Clarke, Park, Phase Voltage Calculation, Sliding Mode Observer, BEMF Commutation, Direct Flux Estimator, Speed Calculators and Estimators, Position Calculators and Estimators etc. |
| **Signal Generators and Control Modules** | PID, Commutation Trigger Generator, V/f Controller, Impulse Generator, Mod 6 Counter, Slew Rate Controllers, Sawtooth & Ramp generators, Space Vector Generators etc. |
| **Peripheral Drivers** | PWM abstraction for multiple topologies and techniques, ADC Drivers, Hall Sensor Driver, QEP Driver, CAP Driver etc. |
| **Real-Time Debugging Modules** | DLOG module for CCS graph window utility, PWMDAC module for monitoring the control variables through socilloscope |

In the DMC library, each module is separately documented with source code, use, and background technical theory. All DMC modules allow users to quickly build, or customize their own systems.

This particular document is all about using resolver interface software library modules. The data types used by the library are presented and the use case description of library functions is outlined.

This document covers the software structure of the Resolver Interface Library. It has a couple of functions and uses a couple of data types to interface to the main project.

The function names are

- Init_resolver_xxx(void)
- Resolver_algo_xxx(void)

and the data types are

- RESOLVER_INPUT
- RESOLVER_OUTPUT

They will be explained in the sections below.

'xxx' in function names represents if it is a CLA function, or FIXED CPU function or FLOAT CPU function. For example, 'init_resolver_Fixed'() is an initializing function used in FIXED CPU based projects.

The available .lib files are given below and are supposed to be included by the linker depending on the CPU / CLA being used to run the resolver functions

| **Library file names** | **Intended Projects Use** |
|---|---|
| Resolver_Lib_fixed.lib | Fixed CPU |
| Resolver_Lib_CLA_fixed.lib | CLA project on Fixed CPU |
| Resolver_Lib_CLA_float.lib | CLA project on Float CPU |
| Resolver_Lib_float.lib | Float CPU |
| Resolver_Lib_float_TMU0.lib | CPUs with TMU (Trigonometric and Math Unit) |

**Data Type Definitions:**

**RESOLVER_INPUT :**

This data type makes it convenient to instance an input interface to resolver library modules. To create multiple instances of the same, simply declare variables of type RESOLVER_INPUT. This structure contains all variables that feed a value into the library, and is given below

```
// Input variables
typedef struct  {
       // variables to set up basic functions
    Uint16 firLag,       // lag between sine index and FIR index
           FIR32,        // select-1 / deselect-0 FIR32 function
           TUNING,       // select-1 / deselect-0 TUNING function
           TABLE_LENGTH; // set up FIR filter length

  float**  offsetS,       // dc offset of sine fbk analog channel
           offsetC,       // dc offset of cosine fbk analog channel
           testAngle,     // test Angle used for tuning the PI coefficients
           SAMPLING_TIME, // loop decimation sampling time (carrier cycle time)

       // control loop parameters (can be replaced with MACROs)
          errorWfT,    // error filter constant, internally computed using filter coefficients
          picon_K0,    // PI controller constant
          picon_K1,    // PI controller constant
          rpsMax;      // max resolver speed in eqvt elec freq
} RESOLVER_INPUT;
```

| Item | Name | Description | Format[*] | Range(Hex) |
|---|---|---|---|---|
| **Flag Inputs** | FIR32 | Select 32 order FIR filter | Q0 | 0000-FFFF |
| | TUNING | Select loop tuning function | Q0 | 0000-FFFF |
| **Parameter Inputs** | firLag | Lag of feedback sample wrt to sine excitation | Q0 | 0000-FFFF |
| | TABLE_LENGTH | Exc Sine table length | Q0 | 0000-FFFF |
| | OffsetS | Residual offset of sine fbk signal | ** | |
| | OffsetC | Residual offset of cosine fbk signal | ** | |
| | testAngle | Angle to test the transient performance of observer | ** | |
| | SAMPLING_TIME | Loop sampling time | ** | |
| | errorWfT | Error filter constant | ** | |
| | picon_k0 | PI controller coefficient | ** | |
| | picon_k1 | PI controller coefficient | ** | |

[*] GLOBAL_Q valued between 1 and 30 is defined in the IQmathLib.h header file.

**These variables are declared as 'float' in floating point CPU projects. Whereas, in fixed point CPU projects, they are declared as int32 or _iq, used in Q20 format, and have a range of 80000000-7FFFFFFF

The names of most variables in the structure are self explanatory. Some of them need a bit explaining to do. 'firLag' is used to position the FIR filter coefficients to appropriate feedback samples from resolver such that the first and last samples of the FIR filter coincide with the peak of excitation carrier wave. This helps to decimate the feedback signals at their max thereby getting higher SNR.

**Availability** C interface version

**Module Properties** **Type:** Target Independent
**Target Devices:** 28x Fixed or Floating Point
**C Version File Names:** resolver_Fixed.h / resolver_CLA.h / resolver_Float.h
**Library files for C:** IQmathLib.h, IQmath.lib // CLAmath.h, CLAmath.lib

## Module Usage

### Instantiation

The following example instance a RESOLVER_INPUT objects
RESOLVER_INPUT rslvrIn;

## Example

The following pseudo code provides the information about the module usage.

```
main()
{
        rslvrIn.FIR32 = 1;  // select 32 order FIR filter
        rslvrIn.TABLE_LENGTH = 32; // excitation sinetable is 32 words long
        .
        .
        .

}
```

**RESOLVER_OUTPUT** :

This data type makes it convenient to instance an output interface to resolver library modules. To create multiple instances of the same, simply declare variables of type RESOLVER_OUTPUT. This structure contains all variables that are output from the library and is given below

```
// Output variables
typedef struct  {
       // variables for outputting results in float
     float** angleRaw,      // raw angle estimate from arctan
             angleObs,      // observer angle estimate w/o FIR delay compensation
             angleOut,      // final angle estimate after FIR delay compensation
             rpsObs,        // shaft speed estimate by the observer
             errorNew,      // new angle error estimated by the observer
             resMag,        // resolver fbk mag

          // debug variables - can be commented out
             sinFIRout,     // FIR band pass filter output of sine feedback signal
             cosFIRout;     // FIR band pass filter output of cosine feedback signal

#ifndef FLOAT_CPU_RESOLVER_CLA_LIB
       // variables for outputting results in Q20
       int32 angleRaw20,    // arctan angle estimate in pu
             angleObs20,    // observer angle estimate in pu
             angleOut20,    // final estimated angle in pu
             rpsObs20,      // shaft speed estimate
             errorNew20,    // PLL loop error in pu

       // variables for data analysis
             resMag20;      // resolver magnitude in Q20
#endif

       // variables used within library
     float** sin_input,    // sine input from resolver
             cos_input;    // cosine input from resolver
       Uint16 sineIndex;   // index to element within sine table
} RESOLVER_OUTPUT;
```

| Item | Name | Description | Format* | Range(Hex) |
|---|---|---|---|---|
| **Observer outputs** | angleRaw | Raw angle estimate | ** | |
| | angleObs | Observer angle estimate | ** | |
| | angleOut | Compensated angle estimate | ** | |
| | rpsObs | Rps estimate by observer | ** | |
| | errorNew | Latest angle error | ** | |
| | resMag | Resolver fbk magnitude | ** | |
| | sinFIRout | Output of sine FIR | ** | |
| | cosFIRout | Output of cosine FIR | ** | |
| | angleRaw20 | Raw angle estimate in Q20 | Q20 | 80000000-7FFFFFFF |
| | angleObs20 | Observer angle estimate in Q20 | Q20 | 80000000-7FFFFFFF |
| | angleOut20 | Compensated angle estimate in Q20 | Q20 | 80000000-7FFFFFFF |
| | rpsObs20 | RPS estimate by observer in Q20 | Q20 | 80000000-7FFFFFFF |
| | errorNew20 | Latest angle error in Q20 | Q20 | 80000000-7FFFFFFF |
| | resMag20 | Resolver fbk magnitude in Q20 | Q20 | 80000000-7FFFFFFF |
| | sin_input | Sine fbk sample from ADC | ** | |
| | cos_input | Cosine fbk sample from ADC | ** | |
| | sineIndex | Index through the exc sine table | Q0 | 0000-FFFF |

*GLOBAL_Q valued between 1 and 30 is defined in the IQmathLib.h header file.

**These variables are declared as ‘float’ in floating point CPU projects. Whereas, in fixed point CPU projects, they are declared as int32 or _iq, used in Q20 format, and have a range of 80000000-7FFFFFFF

The variables suffixing with 20 carry the content in Q20 format for use by a receiving CPU that is fixed point.

Compiler switch `FLOAT_CPU_RESOLVER_CLA_LIB` should be set up if the CPU is a floating point device where _IQ variables are redundant to define.

**Availability** C interface version

**Module Properties** **Type:** Target Independent
**Target Devices:** 28x Fixed or Floating Point
**C Version File Names:** resolver_Fixed.h / resolver_CLA.h / resolver_Float.h
**Library files for C:** IQmathLib.h, IQmath.lib // CLAmath.h, CLAmath.lib

**Module Usage**

**Instantiation**
The following example instance a RESOLVER_OUTPUT objects
RESOLVER_OUTPUT rslvrOut;

**Example**

The following pseudo code provides the information about the module usage.

```
main()
{
        .
        .
        .
        RotorPosition = rslvrOut.angleOut;  // get latest angle from resolver observer
        RotorSpeed    = rslvrIn.rpsObs;       // get latest speed from resolver observer

}
```

**Function Description:**

init_resolver_xxx(void)

- This function initializes the variables used by resolver_algo_xxx()

resolver_algo_xxx(void)

- This function does the FIR band pass filter action and the observer loop as explained in the technical reference document. When a new position data is available, it returns a 1 or else a 0.

**Notes:**
When using CLA lib files

- if CPU is fixed type (such as F28035), use Resolver_Lib_CLA_fixed.lib
- if CPU is float type (such as F28069), use Resolver_Lib_CLA_float.lib

**Availability** C interface version

**Module Properties** **Type:** Target Independent

**Target Devices:** 28x Fixed or Floating Point

**C Version File Names:** resolver_source_CLA.cla // resolver_source_fixed.c // resolver_source_float.c

**Library files for C:** IQmathLib.h, IQmath.lib // CLAmath.h, CLAmath.lib

## IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI’s standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.