iMS4-P Encoder Interface

Velocity / Encoder Compensation Functions

Refer: Isomet Modular Synthesiser (iMS) SDK API documentation
Class: iMS::VelocityConfiguration Struct Reference
(Summarized below page 4. Code Snippet page 6)

iMS4-P synthesisers from Rev-B onwards include a dual optical encoder inputs and built in tracking filters that can be used for example to monitor the velocity of a moving object in two dimensions, compensate the RF frequency by a scaled amount to alter the AOD deflection angle and hence remove smear distortion from the target feature.

Each of the 2 encoder inputs has a pair of RS422 receivers and can be configured to work with both quadrature (for best precision) and clock + direction style encoder signals. The encoder inputs are passed through a glitch filter to remove any excursions < 30ns before being decoded to extract a pulse train and to identify direction of travel.

This information is fed into a tracking loop filter that both attenuates noise from the signal and calculates an estimate for the encoder velocity (in encoder ticks per second). The filter has several parameters that can be adjusted for optimum performance. The transfer function of the filter is:

$$H(s) = \frac{(kp / J.ki).s + 1}{(1 / J.ki).s2 + (kp / J.ki).s + 1}$$

where:

kp = the proportion gain coefficient (CPP= *TrackingLoopProportionCoeff)*

ki = the integral gain coefficient (CPP=*TrackingLoopIntegrationCoeff)*

J = a constant correction factor = 65535 / 687 = 95.393

s = the Laplace operator

The resulting X and Y velocity estimates are applied to the pixel subsystem where they are scaled by a gain coefficient and used to offset the RF channel output frequency from the value requested by Image data, Single Tone or Tone Buffer. The offset is applied as follows:

- If X/Y Phase compensation is enabled (see SDK, *EnableXYPhaseCompensation*), offsets from Encoder input X are applied to RF Channels 1 and 2, offsets from Encoder input Y are applied to RF Channels 3 and 4.

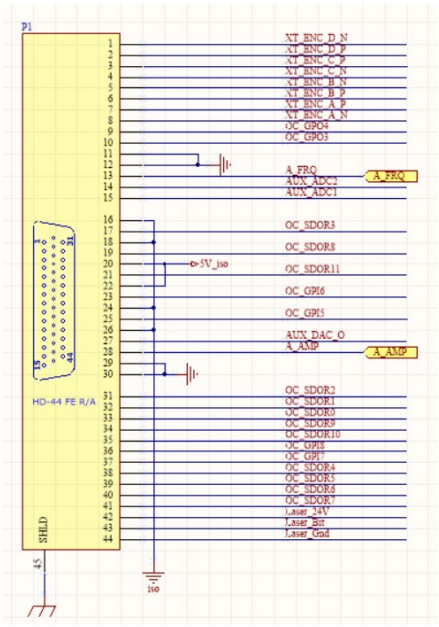
- If X/Y Phase compensation is not enabled, offsets from Encoder input X are applied to all RF Channels and Encoder input Y is ignored.

Note that negative gains are allowed which result in frequency offsets in the opposite direction.

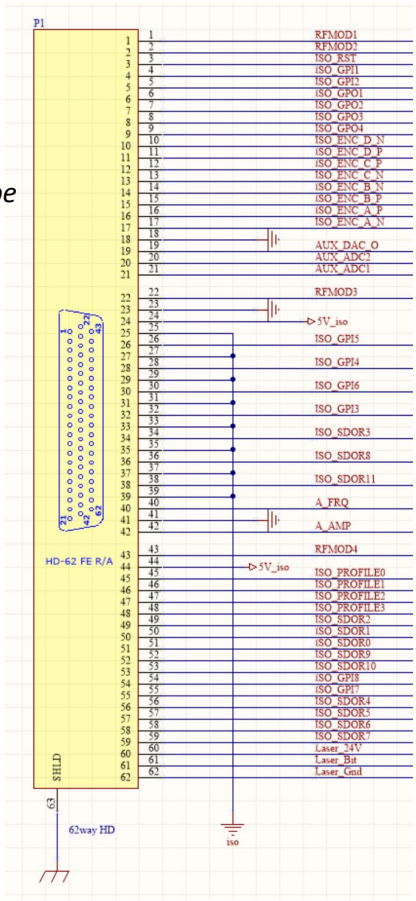
Hardware connections

Connections to the differential quadrature digital encoder inputs are provided on connector J7.

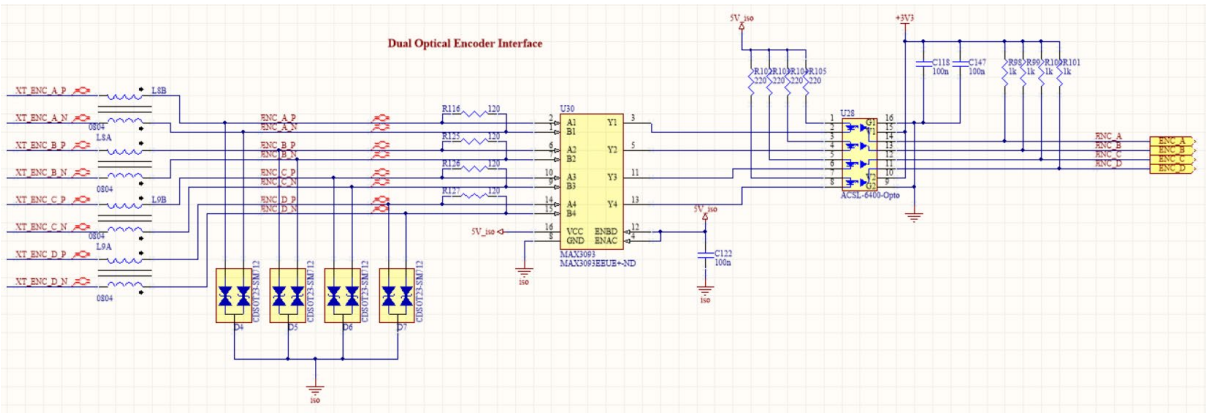
rev-B, rev-C, 44-way D-type



rev-D, 62-way D-type



The interface circuit is detailed below



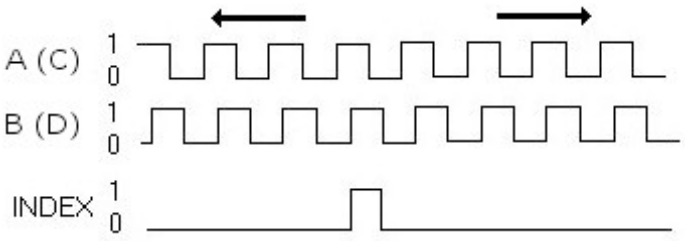
When applied to a 2D scanning system then the digital inputs for a quadrature encoder are assigned as follows:

ENC_A = Cos X-axis
ENC_B = Sin X-axis

ENC_C = Cos Y-axis
ENC_D = Sin Y-axis

For ease of understanding, the input signals are illustrated in simplified form below .

This is a generic digital encoder showing single ended signals.



Although shown above, the iMS4-P does not support Index signal information.
For positional (scan angle) tracking purposes, only the encoder velocity is pertinent . Absolute position information is not required.

The maximum encoder velocity will be limited by the glitch filter (rejects pulses < 30ns) intended to remove mechanical encoder bounce. It will also be affected by the poles and zeros of the tracking filter, which itself will be dependent on the coefficients applied to it by the user through the SDK.

Depending on the filter settings, the upper limit will approach 10MHz per input .
Typical encoder outputs are < 1MHz

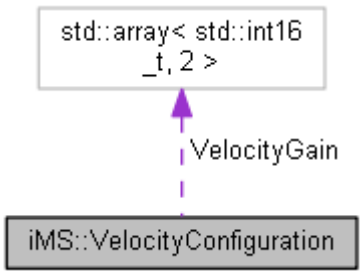
Refer: Isomet Modular Synthesiser (iMS) API classes

iMS::VelocityConfiguration Struct Reference

Sets the parameters required to control the operation of the Encoder Input / Velocity Compensation function. More...

`#include <SignalPath.h>`

Collaboration diagram for iMS::VelocityConfiguration:



[legend]

Public Member Functions

void	SetVelGain (const IMSSystem &ims, SignalPath::ENCODER_CHANNEL chan, kHz EncoderFreq, MHz DesiredFreqDeviation, bool Reverse=false)
	Sets the amount of frequency deviation gain applied to velocity measurement. More...

Public Attributes

SignalPath::ENCODER_MODE	EncoderMode { SignalPath::ENCODER_MODE::QUADRATURE }
	Sets the type of encoder signal connected to the Synthesiser inputs.
SignalPath::VELOCITY_MODE	VelocityMode { SignalPath::VELOCITY_MODE::FAST }
	Sets the velocity calculation method used in the tracking filter for frequency compensation.
std::uint16_t	TrackingLoopProportionCoeff { 4000 }

	The Proportion Coefficient (0 - 65535) used in the Tracking Loop Filter.

std::uint16_t	**TrackingLoopIntegrationCoeff** { 10000 }
	The Integration Coefficient (0 - 65535) used in the Tracking Loop Filter.

std::array< std::int16_t, 2 >	**VelocityGain**
	Controls the extent to which a given value of velocity causes a deviation in synthesiser frequency. Do not set manually, use SetVelGain.

Example CPP code snippet, single tone (calibration) output:

Note: Greyed out areas of code specific to Single tone / Calibration mode set up of iMS4-

```
//
// Somewhere in your source code, create a new class that inherits from the IEventHandler API function
// This will receive data from the iMS when a Encover Velocity readback is requested and print the
// results to the screen
//

    class SignalPathSupervisor : public IEventHandler
    {
    private:
    public:
        void EventAction(void* sender, const int message, const int param)
        {
            switch (message)
            {
            case (SignalPathEvents::ENC_VEL_CH_X) : std::cout << "Encoder Ch X Velocity: " << param << "Hz" << std::endl; break;
            case (SignalPathEvents::ENC_VEL_CH_Y) : std::cout << "Encoder Ch Y Velocity: " << param << "Hz" << std::endl; break;
            }
        }
    };



//
// In your main application code, include the following to set up the encoder and velocity compensation process
// and readback the current velocity every 400ms.
//

// Put these two lines at the top
#include "SignalPath.h"
#include <conio.h>      // for _kbhit()

// Put all the below in your main function, after "USER CODE GOES HERE"

        // Create a SignalPath object
        SignalPath sp(myiMS);

        // Create a supervisor class to respond to readback data and subscribe to velocity events
        SignalPathSupervisor sps;
        sp.SignalPathEventSubscribe(SignalPathEvents::ENC_VEL_CH_X, &sps);
        sp.SignalPathEventSubscribe(SignalPathEvents::ENC_VEL_CH_Y, &sps);

        // Initialise Synthesiser RF with a 70MHz tone
        FAP fap(MHz(70.0), Percent(100.0), Degrees(0.0));
        sp.SetCalibrationTone(fap);

        // Create the default parameters required by the velocity-frequency compensation process
        VelocityConfiguration velcon;
        velcon.VelocityMode = SignalPath::VELOCITY_MODE::SLOW;
        //velcon.TrackingLoopIntegrationCoeff = 10000;  // 0 - 65535. Adjust to suit
        //velcon.TrackingLoopProportionCoeff = 4000;    // 0 - 65535.

        // Set Encoder gain such that a 25kHz encoder tick frequency results in a positive (true) or negative (false)
        // frequency change of 10MHz in channel X (RF Channels 1-4, or in X/Y Phase mode, channels 1-2).
        velcon.SetVelGain(myiMS, SignalPath::ENCODER_CHANNEL::CH_X, kHz(25.0), MHz(10.0), true);

        // Turn on Encoder Input and Velocity-Frequency Compensation process
        sp.UpdateEncoder(velcon);

        do {
            for (int i = 0; i < 10; i++) {

                // A feature of the Synthesiser is that RF outputs are only updated (reflecting a change in encoder velocity and
                    therefore output frequency)whenever a new FAP is issued to the Synthesiser. In Image mode, this happens for
                    each Image point, which will thus be compensated for the current encoder velocity.  In Single Tone Mode (this
                    example snippet), FAPs are only reissued manually through software command.  This loop updates the RF output
                    25 times per second,for human visual purposes.

                std::this_thread::sleep_for(std::chrono::milliseconds(40));
                sp.SetCalibrationTone(fap);
            }

            // Request current velocity
            sp.ReportEncoderVelocity(SignalPath::ENCODER_CHANNEL::CH_X);
        } while(!_kbhit());

        // Use this to turn off the compensation
        sp.DisableEncoder();

        return 0;
```