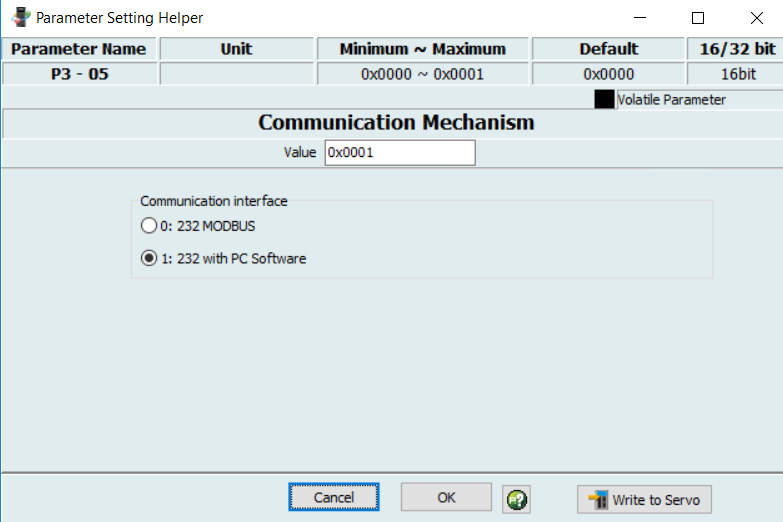
Motor Driver Model: Delta ASDA-B2

POINT:

If you connect the motor driver to adjust a parameter, DO NOT FORGET TO ADJUST P3-05 (COMMUNICATION MECHANISM) PARAMETER TO **0:232 mode bus before you quit the device**. Otherwise our program cannot communicate with the device.

9

POINT:

Mod-bus is master slave protocol. In our code we assumed that the workstation PC is master and the motor driver is slave with address 127 (decimal).

POINTS:

How our Qt-program works (Qt-version 5.8.0):

STEP 1: MAKE CONNECTION OBJECT

// Set connection parameters.

if (device->state() != QModbusDevice::ConnectedState)

{

device->setConnectionParameter(

QModbusDevice::SerialPortNameParameter, "COM3");//port name

device->setConnectionParameter(

QModbusDevice::SerialParityParameter,

QSerialPort::NoParity);

device->setConnectionParameter(

QModbusDevice::SerialBaudRateParameter,

QSerialPort::Baud9600);

device->setConnectionParameter(

QModbusDevice::SerialDataBitsParameter,

QSerialPort::Data8);

device->setConnectionParameter(

QModbusDevice::SerialStopBitsParameter,

QSerialPort::TwoStop);

}

device->setTimeout(1000);

device->setNumberOfRetries(3);

const static qint16 INIT\_SPEED=100;

// now connect to the device

if(device->connectDevice())

{

//QString \_mechineError=device->state();

WriteRegister(1034,INIT\_SPEED);

emit NotifyMachineIsConnected();

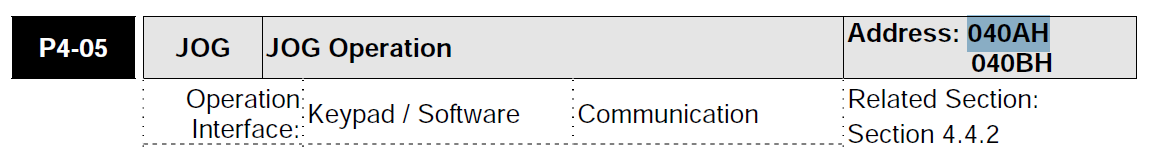
}

Question: What is 1034?

Answer: It is the address of a register in decimal which is going to be written in. In this example, I know that after successful connection I need to initialize the initial motors rotatıon speed. Otherwise the device may initialized into a very high rotation speed and any start command may cause a serious damage. So it is safe to initialize the rotation speed.

Question: How did you get that?

Answer: First I have downloaded the DELTA ASDA B2 Manual (<http://www.delta.com.tw/product/em/motion/motion_servo/download/manual/DELTA_ASDA-B2_M_EN_20130906.pdf>), then I find the speed register at chapter seven (servo parameters)



As you can see 040AH and 040BH are two bytes that jog speed is written into. As far as INIT\_SPEED is 16-bits data register (i.e. qint16) so the types matches with each other. I take the initial address (040A in Hex) and convert it into decimal value (using Microsoft calculator in programmer mode):



Same procedure is applied for the rest of parameters:

STEP 2: Turn on the servo motor

const static int SLAVE\_ADDRESS = 127;

// write servo-on register on P2-30 (1: turns on & 0: turns off)

QVector<quint16> writeBuffer(1);

writeBuffer[0] = value;

QModbusDataUnit adu( QModbusDataUnit::HoldingRegisters, **572**, writeBuffer);

if (auto \*reply = device->sendWriteRequest(adu, SLAVE\_ADDRESS))

{

if (!reply->isFinished())

{

connect( reply, &QModbusReply::finished, this,

&Machinary::writeReady);

}

else

{

delete reply; // broadcast replies return immediately

}

}

else

{

QString \_errorMsg=device->errorString();

emit NotifyErrorHappened(\_errorMsg);

}

void Machinary::writeReady()

{

auto reply = qobject\_cast<QModbusReply \*>(sender());

if (!reply)

return;

if (reply->error() == QModbusDevice::ProtocolError)

{

emit NotifyErrorHappened(\_errorMsg);

}

else if (reply->error() != QModbusDevice::NoError)

{

emit NotifyErrorHappened(\_errorMsg);

}

reply->deleteLater();

}

STEP 3: Start and Stop moving

For this we need to invoke Jog operation. The delta motor driver is working in this way.

* If you write 4999 into **P4-05** and the servo is on (i.e. **P2-30 is 1**), then motor start to move in *forward* direction.
* If you write 4998 into **P4-05** and the servo is on (i.e. **P2-30 is 1**), then motor start to move in *backward* direction.
* If you write 0 into P4-05 and the servo is on, then motor *stop* from moving but it still remains on (you can hear its gizz sound!).
* If you write any other value between 1 and 5000 into P4-05 it will be the used as *rotation speed* [r/min].

Point: if you want to change the speed while the motor still is moving, then you need to first update the speed then immediately re-submit the move forward/backward command.

WriteRegister(1034,value);

ReadRegister(1034);

switch (moveStat) {

case MOVE\_STAT::FORWARD:

MoveForward();

break;

case MOVE\_STAT::BACKWARD:

MoveBackWard();

break;

case MOVE\_STAT::STOP:

StopMachine();

break;

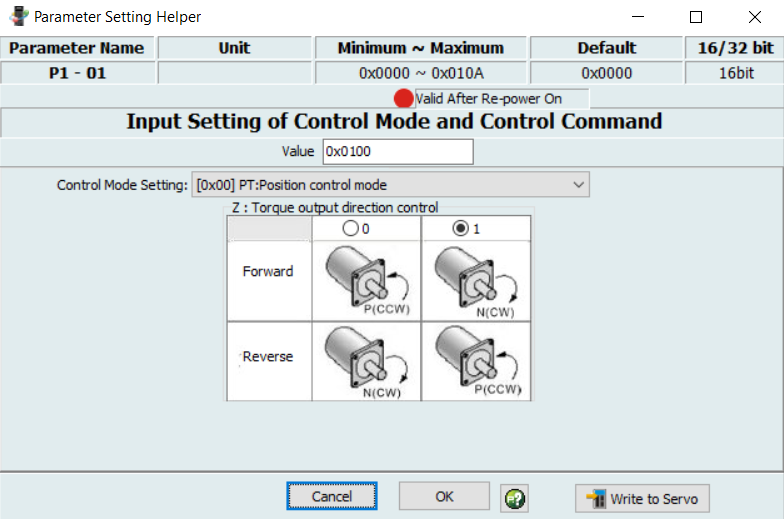
default:

break;

}

WriteRegister, MoveForward , MoveBackWard are available in source code. They are not part of Qt mod-bus.

The servo motor works in six single and five dual mods (Delta servo motor manual chapter 6 page 116). In our case the device actuator is designed to work in position control mode. You can select the mode at Parameter P1-01.

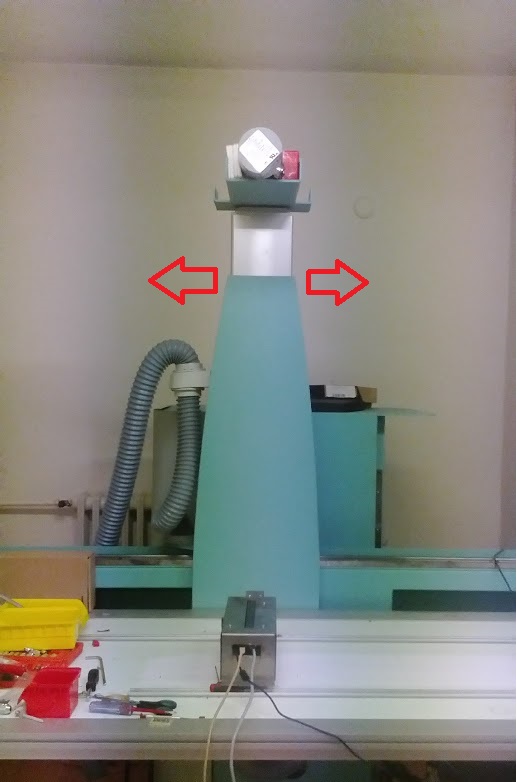


The control diagram in this case is as follow.

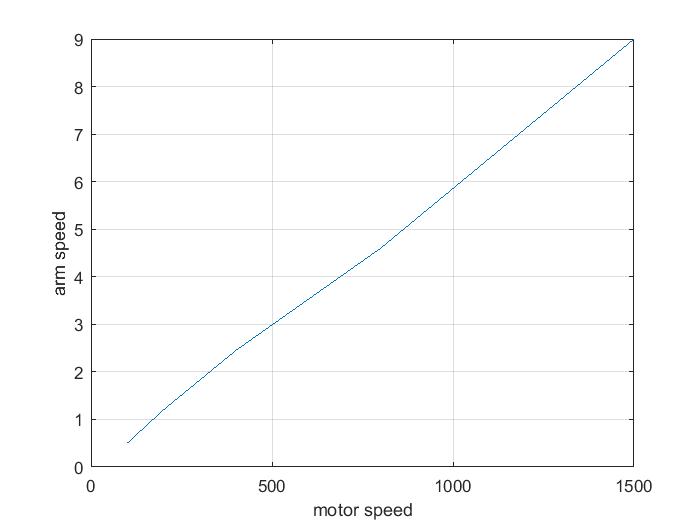


In our device, there is a PLC that generate suitable timing signals and handles the issues. As we have no access to the PLC program and also we have no idea about the wiring between PLC and motor driver, we just accept the default settings and do not change them.

But in our application we need to study the relation between detector speed (i.e. line speed in Hz) and the actuator speed (arm speed). As we have said the arm speed depends on the rotation speed (P4-05). I noticed that there is a linear dependency between rotation speed and the arm speed.



|  |  |  |  |
| --- | --- | --- | --- |
| Rotation Speed [r/min] | Distance [cm] | Duration [sec] | Speed [cm/sec] |
| 100 | 5 | 10 | 0.5 |
| 200 | 12 | 10 | 1.2 |
| 400 | 24.5 | 10 | 2.45 |
| 600 | 17.5 | 5 | 3.5 |
| 800 | 23 | 5 | 4.6 |
| 1500 | 45 | 5 | 9 |
| 2000 | 57 | 5 | 11.4 |



Detector

The following code is from sample come from Teledyne detector installed in “C:\Program Files (x86)\Teledyne Dalsa\ArgusAPI\Example”. I was trying to acquire images, at

/\* grab image \*/

strcpy\_s( Cmd, "acq" );

ret = \_camera\_cmd( Cmd, \_cam\_info.CameraResp, CmdTimeout );

//printf("%s\n", \_cam\_info.CameraResp);

if(ret != 0)

{

return ret;

}

I put a break point but I did not let the program to complete the acquisition process. I found a serious collapse that does not let the program to recompile so I had to restart visual studio. I found the following section is a requirement

/\* Close Camera operation \*/

**Sleep(3); /\* Fast connect and disconnect needs some time gap other, communication is prone to break \*/**

ret = \_camera\_disconnect( &\_cam\_info.WSAStatus, \_cam\_info.CameraResp, CmdTimeout );

if(ret != 0)

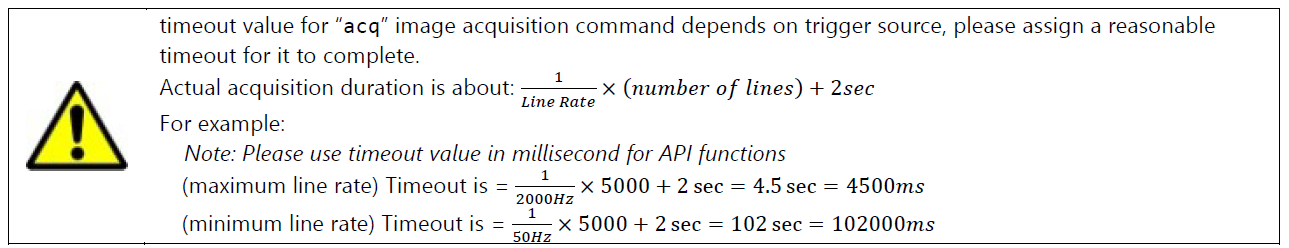
{

return ret;

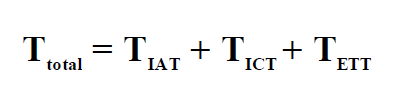
}

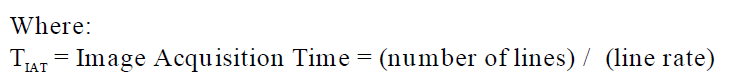
So, always try to finish the "acq" command or wait for its timeout. If any emergency stop commands arrived then you should prevent immediate acquisition and must wait for a little while.

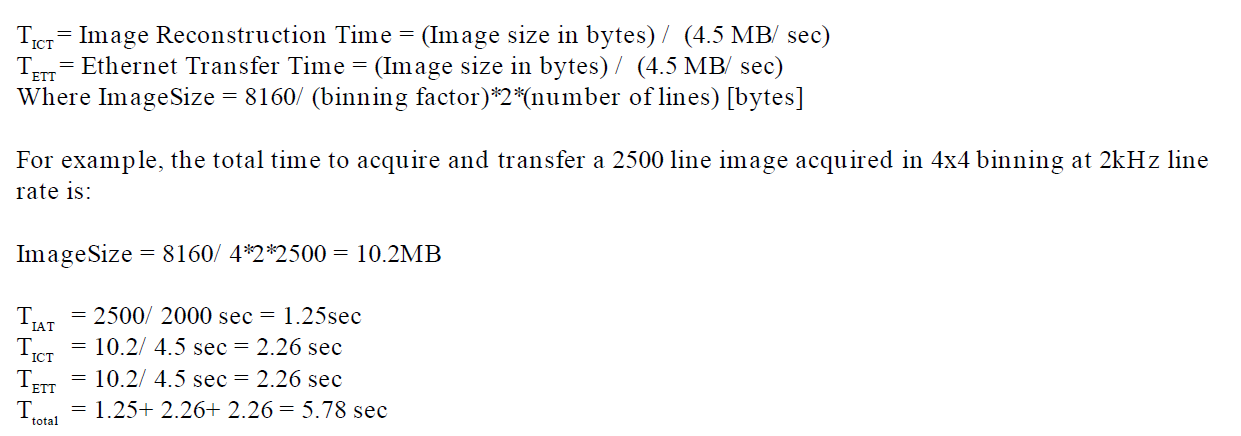
There are two timeout formulas required for “acq” command:



And







In my experiments I noticed the second formula is far more accurate. So, do not take wrong timeout.

Dilemma:

I want to take [**c**] cm image in [**t**] second time interval.

What is the “*line speed rate*” of detector?

What the best “*motor rotation speed*”?