Measurement Computing 1608GX-2A0 Driver

Version 1

* 2 simple analog outputs
* 1 parameter, ANALOG\_OUT\_VALUE
* Use asynPortDriver C++ base class
* 131 lines of code

/\* drvUSB1608G\_V1.cpp

 \*

 \* Driver for Measurement Computing USB-1608G

 \* multi-function DAQ board using asynPortDriver base class

 \*

 \* This version implements only simple analog outputs

 \*

 \* Mark Rivers

 \* April 14, 2012

\*/

#include <iocsh.h>

#include <epicsExport.h>

#include <asynPortDriver.h>

#include "cbw.h"

static const char \*driverName = "USB1608G";

// Analog output parameters

#define analogOutValueString "ANALOG\_OUT\_VALUE"

#define NUM\_ANALOG\_OUT 2 // Number of analog outputs on 1608G

#define MAX\_SIGNALS NUM\_ANALOG\_OUT

/\*\* Class definition for the USB1608G class

 \*/

class USB1608G : public asynPortDriver {

public:

 USB1608G(const char \*portName, int boardNum);

 /\* These are the methods that we override from asynPortDriver \*/

 virtual asynStatus writeInt32(asynUser \*pasynUser, epicsInt32 value);

 virtual asynStatus getBounds(asynUser \*pasynUser, epicsInt32 \*low,

 epicsInt32 \*high);

protected:

 // Analog output parameters

 int analogOutValue\_;

 #define FIRST\_USB1608G\_PARAM analogOutValue\_

 #define LAST\_USB1608G\_PARAM analogOutValue\_

private:

 int boardNum\_;

};

#define NUM\_PARAMS (&LAST\_USB1608G\_PARAM - &FIRST\_USB1608G\_PARAM + 1)

/\*\* Constructor for the USB1608G class

 \*/

USB1608G::USB1608G(const char \*portName, int boardNum)

 : asynPortDriver(portName, MAX\_SIGNALS, NUM\_PARAMS,

 asynInt32Mask | asynDrvUserMask, // Interfaces that we implement

 0, // Interfaces that do callbacks

 ASYN\_MULTIDEVICE | ASYN\_CANBLOCK, 1,

 /\* ASYN\_CANBLOCK=1, ASYN\_MULTIDEVICE=1, autoConnect=1 \*/

 0, 0), /\* Default priority and stack size \*/

 boardNum\_(boardNum)

{

 // Analog output parameters

 createParam(analogOutValueString, asynParamInt32, &analogOutValue\_);

}

asynStatus USB1608G::getBounds(asynUser \*pasynUser, epicsInt32 \*low,

 epicsInt32 \*high)

{

 int function = pasynUser->reason;

 // Analog outputs are 16-bit devices

 if (function == analogOutValue\_) {

 \*low = 0;

 \*high = 65535;

 return(asynSuccess);

 } else {

 return(asynError);

 }

}

asynStatus USB1608G::writeInt32(asynUser \*pasynUser, epicsInt32 value)

{

 int addr;

 int function = pasynUser->reason;

 int status=0;

 static const char \*functionName = "writeInt32";

 this->getAddress(pasynUser, &addr);

 setIntegerParam(addr, function, value);

 // Analog output functions

 if (function == analogOutValue\_) {

 status = cbAOut(boardNum\_, addr, BIP10VOLTS, value);

 }

 callParamCallbacks(addr);

 if (status == 0) {

 asynPrint(pasynUser, ASYN\_TRACEIO\_DRIVER,

 "%s:%s, port %s, wrote %d to address %d\n",

 driverName, functionName, this->portName, value, addr);

 } else {

 asynPrint(pasynUser, ASYN\_TRACE\_ERROR,

 "%s:%s, port %s, ERROR writing %d to address %d, status=%d\n",

 driverName, functionName, this->portName, value, addr, status);

 }

 return (status==0) ? asynSuccess : asynError;

}

/\*\* Configuration command, called directly or from iocsh \*/

extern "C" int USB1608GConfig(const char \*portName, int boardNum)

{

 USB1608G \*pUSB1608G = new USB1608G(portName, boardNum);

 pUSB1608G = NULL; /\* This is just to avoid compiler warnings \*/

 return(asynSuccess);

}

static const iocshArg configArg0 = { "Port name", iocshArgString};

static const iocshArg configArg1 = { "Board number", iocshArgInt};

static const iocshArg \* const configArgs[] = {&configArg0,

 &configArg1};

static const iocshFuncDef configFuncDef = {"USB1608GConfig", 2, configArgs};

static void configCallFunc(const iocshArgBuf \*args)

{

 USB1608GConfig(args[0].sval, args[1].ival);

}

void drvUSB1608GRegister(void)

{

 iocshRegister(&configFuncDef,configCallFunc);

}

extern "C" {

epicsExportRegistrar(drvUSB1608GRegister);

}

**measCompAnalogOut.template**

**record(ao,"$(P)$(R)") {**

 **field(PINI, "YES")**

 **field(DTYP, "asynInt32")**

 **field(FLNK, "$(P)$(R)Return.PROC PP MS")**

 **field(OUT, "@asyn($(PORT),$(ADDR))ANALOG\_OUT\_VALUE")**

 **field(EGUL, "$(EGUL)")**

 **field(DRVL, "$(DRVL)")**

 **field(LOPR, "$(LOPR)")**

 **field(EGUF, "$(EGUF)")**

 **field(DRVH, "$(DRVH)")**

 **field(HOPR, "$(HOPR)")**

 **field(PREC, "$(PREC)")**

 **field(LINR, "LINEAR")**

 **field(VAL, "$(VAL)")**

**}**

**record(ao,"$(P)$(R)Return") {**

 **field(DTYP, "asynInt32")**

 **field(DISV, "0")**

 **field(SDIS, "$(P)$(R)Pulse.VAL NPP NMS")**

 **field(OUT, "@asyn($(PORT),$(ADDR))ANALOG\_OUT\_VALUE")**

 **field(OMSL, "closed\_loop")**

 **field(EGUL, "$(EGUL)")**

 **field(DRVL, "$(DRVL)")**

 **field(LOPR, "$(LOPR)")**

 **field(EGUF, "$(EGUF)")**

 **field(DRVH, "$(DRVH)")**

 **field(HOPR, "$(HOPR)")**

 **field(PREC, "$(PREC)")**

 **field(LINR, "LINEAR")**

 **field(VAL, "$(VAL)")**

**}**

**record(bo,"$(P)$(R)Pulse") {**

 **field(ZNAM, "Normal")**

 **field(ONAM, "Pulse")**

**}**

**record(ao,"$(P)$(R)TweakVal") {**

 **field(PREC, "$(PREC)")**

**}**

**record(calcout,"$(P)$(R)TweakUp") {**

 **field(CALC, "A+B")**

 **field(INPA, "$(P)$(R).VAL NPP MS")**

 **field(INPB, "$(P)$(R)TweakVal.VAL NPP MS")**

 **field(OUT, "$(P)$(R).VAL PP MS")**

 **field(PREC, "$(PREC)")**

**}**

**record(calcout,"$(P)$(R)TweakDown") {**

 **field(CALC, "A-B")**

 **field(INPA, "$(P)$(R).VAL NPP MS")**

 **field(INPB, "$(P)$(R)TweakVal.VAL NPP MS")**

 **field(OUT, "$(P)$(R).VAL PP MS")**

 **field(PREC, "$(PREC)")**

**}**

**st.cmd\_V1**

**< envPaths**

**## Register all support components**

**dbLoadDatabase "../../dbd/measCompApp.dbd"**

**measCompApp\_registerRecordDeviceDriver pdbbase**

**dbLoadTemplate("1608G.substitutions\_V1")**

**## Configure port driver**

**# USB1608GConfig(**

**# portName, # The name to give to this asyn port driver**

**# boardNum) # The number of this board assigned by the**

 **# Measurement Computing Instacal program**

**USB1608GConfig("1608G\_1", 1)**

**#asynSetTraceMask 1608G\_1 -1 255**

**iocInit**

**1608G.substitutions\_V1**

**# asyn record**

**file "$(ASYN)/db/asynRecord.db"**

**{**

**pattern**

**{ P, R, PORT, ADDR, IMAX, OMAX}**

**{ 1608G:, asyn1, 1608G\_1, 0, 80, 80}**

**}**

**# Analog outputs**

**file "$(MEASCOMP)/measCompApp/Db/measCompAnalogOut.template"**

**{**

**pattern**

**{ P, R, VAL, PORT, ADDR, EGUL, DRVL, LOPR, EGUF, DRVH, HOPR, PREC}**

**{ 1608G:, Ao1, 0, 1608G\_1, 0, -10., -10., -10., 10., 10., 10., 4}**

**{ 1608G:, Ao2, 0, 1608G\_1, 1, -10., -10., -10., 10., 10., 10., 4}**

**}**



Measurement Computing 1608GX-2A0 Driver

Version 2

* Add 8 simple analog inputs
	+ 2 new parameters, ANALOG\_IN\_VALUE, ANALOG\_IN\_RANGE
* Add report() function
* 197 lines of code (~60 more than V1)

/\* drvUSB1608G\_V2.cpp

 \*

 \* Driver for Measurement Computing USB-1608G multi-function DAQ board using asynPortDriver base class

 \*

 **\* This version implements simple analog inputs and simple analog outputs**

 \*

 \* Mark Rivers

 \* April 14, 2012

\*/

#include <iocsh.h>

#include <epicsExport.h>

#include <asynPortDriver.h>

#include "cbw.h"

static const char \*driverName = "USB1608G";

// Analog output parameters

#define analogOutValueString "ANALOG\_OUT\_VALUE"

**// Analog input parameters**

**#define analogInValueString "ANALOG\_IN\_VALUE"**

**#define analogInRangeString "ANALOG\_IN\_RANGE"**

**#define NUM\_ANALOG\_IN 16 // Number analog inputs on 1608G**

#define NUM\_ANALOG\_OUT 2 // Number of analog outputs on 1608G

**#define MAX\_SIGNALS NUM\_ANALOG\_IN**

/\*\* Class definition for the USB1608G class

 \*/

class USB1608G : public asynPortDriver {

public:

 USB1608G(const char \*portName, int boardNum);

 /\* These are the methods that we override from asynPortDriver \*/

 virtual asynStatus writeInt32(asynUser \*pasynUser, epicsInt32 value);

 **virtual asynStatus readInt32(asynUser \*pasynUser, epicsInt32 \*value);**

virtual asynStatus getBounds(asynUser \*pasynUser, epicsInt32 \*low,

 epicsInt32 \*high);

 **virtual void report(FILE \*fp, int details);**

protected:

 // Analog output parameters

 int analogOutValue\_;

 #define FIRST\_USB1608G\_PARAM analogOutValue\_

 **// Analog input parameters**

 **int analogInValue\_;**

 **int analogInRange\_;**

 **#define LAST\_USB1608G\_PARAM analogInRange\_**

private:

 int boardNum\_;

};

#define NUM\_PARAMS (&LAST\_USB1608G\_PARAM - &FIRST\_USB1608G\_PARAM + 1)

/\*\* Constructor for the USB1608G class

 \*/

USB1608G::USB1608G(const char \*portName, int boardNum)

 : asynPortDriver(portName, MAX\_SIGNALS, NUM\_PARAMS,

 asynInt32Mask | asynDrvUserMask, // Interfaces that we implement

 0, // Interfaces that do callbacks

 ASYN\_MULTIDEVICE | ASYN\_CANBLOCK, 1,

 /\* ASYN\_CANBLOCK=1, ASYN\_MULTIDEVICE=1, autoConnect=1 \*/

 0, 0), /\* Default priority and stack size \*/

 boardNum\_(boardNum)

{

 // Analog output parameters

 createParam(analogOutValueString, asynParamInt32, &analogOutValue\_);

 **// Analog input parameters**

 **createParam(analogInValueString, asynParamInt32, &analogInValue\_);**

 **createParam(analogInRangeString, asynParamInt32, &analogInRange\_);**

}

asynStatus USB1608G::getBounds(asynUser \*pasynUser, epicsInt32 \*low,

 epicsInt32 \*high)

{

 int function = pasynUser->reason;

 **// Both the analog outputs and analog inputs are 16-bit devices**

 **if ((function == analogOutValue\_) ||**

 **(function == analogInValue\_)) {**

 \*low = 0;

 \*high = 65535;

 return(asynSuccess);

 } else {

 return(asynError);

 }

}

 **NOTE: No change, we don’t handle analogOutRange\_ here,**

**just put in parameter library**

asynStatus USB1608G::writeInt32(asynUser \*pasynUser, epicsInt32 value)

{

 int addr;

 int function = pasynUser->reason;

 int status=0;

 static const char \*functionName = "writeInt32";

 this->getAddress(pasynUser, &addr);

 setIntegerParam(addr, function, value);

 // Analog output functions

 if (function == analogOutValue\_) {

 status = cbAOut(boardNum\_, addr, BIP10VOLTS, value);

 }

 callParamCallbacks(addr);

 if (status == 0) {

 asynPrint(pasynUser, ASYN\_TRACEIO\_DRIVER,

 "%s:%s, port %s, wrote %d to address %d\n",

 driverName, functionName, this->portName, value, addr);

 } else {

 asynPrint(pasynUser, ASYN\_TRACE\_ERROR,

 "%s:%s, port %s, ERROR writing %d to address %d, status=%d\n",

 driverName, functionName, this->portName, value, addr, status);

 }

 return (status==0) ? asynSuccess : asynError;

}

**asynStatus USB1608G::readInt32(asynUser \*pasynUser, epicsInt32 \*value)**

**{**

 **int addr;**

 **int function = pasynUser->reason;**

 **int status=0;**

 **unsigned short shortVal;**

 **int range;**

 **//static const char \*functionName = "readInt32";**

 **this->getAddress(pasynUser, &addr);**

 **// Analog input function**

 **if (function == analogInValue\_) {**

 **getIntegerParam(addr, analogInRange\_, &range);**

 **status = cbAIn(boardNum\_, addr, range, &shortVal);**

 **\*value = shortVal;**

 **setIntegerParam(addr, analogInValue\_, \*value);**

 **}**

 **// Other functions we call the base class method**

 **else {**

 **status = asynPortDriver::readInt32(pasynUser, value);**

 **}**

 **callParamCallbacks(addr);**

 **return (status==0) ? asynSuccess : asynError;**

**}**

**/\* Report parameters \*/**

**void USB1608G::report(FILE \*fp, int details)**

**{**

 **int i;**

 **int range;**

 **fprintf(fp, " Port: %s, board number=%d\n",**

 **this->portName, boardNum\_);**

 **if (details >= 1) {**

 **for (i=0; i<NUM\_ANALOG\_IN; i++) {**

 **getIntegerParam(i, analogInRange\_, &range);**

 **fprintf(fp, "channel %d range=%d", i, range);**

 **}**

 **fprintf(fp, "\n");**

 **}**

 **asynPortDriver::report(fp, details);**

**}**

**measCompAnalogIn.template**

**record(ai,"$(P)$(R)")**

**{**

 **field(SCAN, "$(SCAN)")**

 **field(DTYP, "asynInt32")**

 **field(INP, "@asyn($(PORT),$(ADDR))ANALOG\_IN\_VALUE")**

 **field(LINR, "LINEAR")**

 **field(EGUF, "$(EGUF)")**

 **field(EGUL, "$(EGUL)")**

 **field(HOPR, "$(HOPR)")**

 **field(LOPR, "$(LOPR)")**

 **field(PREC, "$(PREC)")**

**}**

**# Note: the ZRVL, etc. fields correspond to the values of BIP1VOLTS, etc. in cbw.h**

**record(mbbo,"$(P)$(R)Range")**

**{**

 **field(PINI, "YES")**

 **field(DTYP, "asynInt32")**

 **field(OUT, "@asyn($(PORT),$(ADDR))ANALOG\_IN\_RANGE")**

 **field(ZRST, "+-1V")**

 **field(ZRVL, "4")**

 **field(ONST, "+-2V")**

 **field(ONVL, "14")**

 **field(TWST, "+-5V")**

 **field(TWVL, "0")**

 **field(THST, "+-10V")**

 **field(THVL, "1")**

 **field(VAL, "$(RANGE)")**

**}**

**1608G.substitutions\_V2**

**# Analog inputs**

**file "$(MEASCOMP)/measCompApp/Db/measCompAnalogIn.template"**

**{**

**pattern**

**{ P, R, PORT, ADDR, EGUL, LOPR, EGUF, HOPR, RANGE, SCAN, PREC}**

**{ 1608G:, Ai1, 1608G\_1, 0, -10., -10., 10., 10., 3, "1 second", 4}**

**{ 1608G:, Ai2, 1608G\_1, 1, -10., -10., 10., 10., 3, "1 second", 4}**

**{ 1608G:, Ai3, 1608G\_1, 2, -5., -5., 5., 5., 2, "1 second", 4}**

**{ 1608G:, Ai4, 1608G\_1, 3, -5., -5., 5., 5., 2, "1 second", 4}**

**{ 1608G:, Ai5, 1608G\_1, 4, -2., -2., 2., 2., 1, "1 second", 4}**

**{ 1608G:, Ai6, 1608G\_1, 5, -2., -2., 2., 2., 1, "1 second", 4}**

**{ 1608G:, Ai7, 1608G\_1, 6, -1., -1., 1., 1., 0, "1 second", 4}**

**{ 1608G:, Ai8, 1608G\_1, 7, -1., -1., 1., 1., 0, "1 second", 4}**

**}**

Measurement Computing 1608GX-2A0 Driver

Version 3

* Add digital outputs
	+ 2 new parameters, DIGITAL\_DIRECTION, DIGITAL\_OUTPUT
* 253 lines of code (~60 more than Version 2)

/\* drvUSB1608G\_V3.cpp

 \*

 \* This version implements digital outputs, simple analog inputs and simple analog outputs

 \*

 \* Mark Rivers

 \* April 14, 2012

\*/

#include <iocsh.h>

#include <epicsExport.h>

#include <asynPortDriver.h>

#include "cbw.h"

static const char \*driverName = "USB1608G";

// Analog output parameters

#define analogOutValueString "ANALOG\_OUT\_VALUE"

// Analog input parameters

#define analogInValueString "ANALOG\_IN\_VALUE"

#define analogInRangeString "ANALOG\_IN\_RANGE"

**// Digital I/O parameters**

**#define digitalDirectionString "DIGITAL\_DIRECTION"**

**#define digitalOutputString "DIGITAL\_OUTPUT"**

#define NUM\_ANALOG\_IN 16 // Number analog inputs on 1608G

#define NUM\_ANALOG\_OUT 2 // Number of analog outputs on 1608G

**#define NUM\_IO\_BITS 8 // Number of digital I/O bits on 1608G**

#define MAX\_SIGNALS NUM\_ANALOG\_IN

/\*\* Class definition for the USB1608G class

 \*/

class USB1608G : public asynPortDriver {

public:

 USB1608G(const char \*portName, int boardNum);

 /\* These are the methods that we override from asynPortDriver \*/

 virtual asynStatus writeInt32(asynUser \*pasynUser, epicsInt32 value);

 virtual asynStatus readInt32(asynUser \*pasynUser, epicsInt32 \*value);

virtual asynStatus getBounds(asynUser \*pasynUser, epicsInt32 \*low,

 epicsInt32 \*high);

**virtual asynStatus writeUInt32Digital(asynUser \*pasynUser, epicsUInt32 value,**

 **epicsUInt32 mask);**

 virtual void report(FILE \*fp, int details);

protected:

 // Analog output parameters

 int analogOutValue\_;

 #define FIRST\_USB1608G\_PARAM analogOutValue\_

 // Analog input parameters

 int analogInValue\_;

 int analogInRange\_;

 **// Digital I/O parameters**

 **int digitalDirection\_;**

 **int digitalOutput\_;**

 #define LAST\_USB1608G\_PARAM digitalOutput\_

private:

 int boardNum\_;

};

/\*\* Constructor for the USB1608G class

 \*/

USB1608G::USB1608G(const char \*portName, int boardNum)

 : asynPortDriver(portName, MAX\_SIGNALS, NUM\_PARAMS,

 // Interfaces that we implement

 asynInt32Mask | **asynUInt32DigitalMask** | asynDrvUserMask,

 **// Interfaces that do callbacks**

 **asynUInt32DigitalMask,**

 ASYN\_MULTIDEVICE | ASYN\_CANBLOCK, 1,

 /\* ASYN\_CANBLOCK=1, ASYN\_MULTIDEVICE=1, autoConnect=1 \*/

 0, 0), /\* Default priority and stack size \*/

 boardNum\_(boardNum)

{

 // Analog output parameters

 createParam(analogOutValueString, asynParamInt32, &analogOutValue\_);

 // Analog input parameters

 createParam(analogInValueString, asynParamInt32, &analogInValue\_);

 createParam(analogInRangeString, asynParamInt32, &analogInRange\_);

 **// Digital I/O parameters**

 **createParam(digitalDirectionString, asynParamUInt32Digital, &digitalDirection\_);**

 **createParam(digitalOutputString, asynParamUInt32Digital, &digitalOutput\_);**

}

**asynStatus USB1608G::writeUInt32Digital(asynUser \*pasynUser, epicsUInt32 value, epicsUInt32 mask)**

**{**

 **int function = pasynUser->reason;**

 **int status=0;**

 **int i;**

 **epicsUInt32 outValue=0, outMask, direction=0;**

 **static const char \*functionName = "writeUInt32Digital";**

 **setUIntDigitalParam(function, value, mask);**

 **if (function == digitalDirection\_) {**

 **outValue = (value == 0) ? DIGITALIN : DIGITALOUT;**

 **for (i=0; i<NUM\_IO\_BITS; i++) {**

 **if ((mask & (1<<i)) != 0) {**

 **status = cbDConfigBit(boardNum\_, AUXPORT, i, outValue);**

 **}**

 **}**

 **}**

 **else if (function == digitalOutput\_) {**

 **getUIntDigitalParam(digitalDirection\_, &direction, 0xFFFFFFFF);**

 **for (i=0, outMask=1; i<NUM\_IO\_BITS; i++, outMask = (outMask<<1)) {**

 **// Only write the value if the mask has this bit set and the direction**

 **// for that bit is output (1)**

 **outValue = ((value &outMask) == 0) ? 0 : 1;**

 **if ((mask & outMask & direction) != 0) {**

 **status = cbDBitOut(boardNum\_, AUXPORT, i, outValue);**

 **}**

 **}**

 **}**

 **callParamCallbacks();**

 **if (status == 0) {**

 **asynPrint(pasynUser, ASYN\_TRACEIO\_DRIVER,**

 **"%s:%s, port %s, wrote outValue=0x%x, value=0x%x, mask=0x%x, direction=0x%x\n",**

 **driverName, functionName, this->portName, outValue, value, mask, direction);**

 **} else {**

 **asynPrint(pasynUser, ASYN\_TRACE\_ERROR,**

 **"%s:%s, port %s, ERROR writing outValue=0x%x, value=0x%x, mask=0x%x,**

 **direction=0x%x, status=%d\n",**

 **driverName, functionName, this->portName, outValue, value, mask, direction,**

 **status);**

 **}**

 **return (status==0) ? asynSuccess : asynError;**

**}**

**measCompBinaryDir.template**

**record(bo, "$(P)$(R)")**

**{**

 **field(PINI, "YES")**

 **field(DTYP, "asynUInt32Digital")**

 **field(OUT, "@asynMask($(PORT),$(ADDR),$(MASK))DIGITAL\_DIRECTION")**

 **field(ZNAM, "In")**

 **field(ONAM, "Out")**

 **field(VAL, "$(VAL)")**

 **field(PINI, "YES")**

**}**

**measCompBinaryOut.template**

**record(bo, "$(P)$(R)")**

**{**

 **field(PINI, "YES")**

 **field(DTYP, "asynUInt32Digital")**

 **field(OUT, "@asynMask($(PORT),$(ADDR),$(MASK))DIGITAL\_OUTPUT")**

 **field(ZNAM, "Low")**

 **field(ONAM, "High")**

**}**

**# This is a readback of the output, with SCAN=I/O Intr**

**record(bi, "$(P)$(R)\_RBV")**

**{**

 **field(DTYP, "asynUInt32Digital")**

 **field(INP, "@asynMask($(PORT),$(ADDR),$(MASK))DIGITAL\_OUTPUT")**

 **field(ZNAM, "Low")**

 **field(ONAM, "High")**

 **field(SCAN, "I/O Intr")**

**}**

**measCompLongOut.template**

**record(longout, "$(P)$(R)")**

**{**

 **field(PINI, "YES")**

 **field(DTYP, "asynUInt32Digital")**

 **field(OUT, "@asynMask($(PORT),$(ADDR),$(MASK))DIGITAL\_OUTPUT")**

**}**

**# This is a readback of the output, with SCAN=I/O Intr**

**record(longin, "$(P)$(R)\_RBV")**

**{**

 **field(DTYP, "asynUInt32Digital")**

 **field(INP, "@asynMask($(PORT),$(ADDR),$(MASK))DIGITAL\_OUTPUT")**

 **field(SCAN, "I/O Intr")**

**}**

**1608G.substitutions\_V3**

**file "$(MEASCOMP)/measCompApp/Db/measCompLongOut.template"**

**{**

**pattern**

**{ P, R, MASK, PORT, ADDR}**

**{ 1608G:, Lo, 0xFF, 1608G\_1, 0}**

**}**

**file "$(MEASCOMP)/measCompApp/Db/measCompBinaryOut.template"**

**{**

**pattern**

**{ P, R, MASK, PORT, ADDR}**

**{ 1608G:, Bo1, 0x01 1608G\_1, 0}**

**{ 1608G:, Bo2, 0x02 1608G\_1, 0}**

**{ 1608G:, Bo3, 0x04 1608G\_1, 0}**

**{ 1608G:, Bo4, 0x08 1608G\_1, 0}**

**{ 1608G:, Bo5, 0x10 1608G\_1, 0}**

**{ 1608G:, Bo6, 0x20 1608G\_1, 0}**

**{ 1608G:, Bo7, 0x40 1608G\_1, 0}**

**{ 1608G:, Bo8, 0x80 1608G\_1, 0}**

**}**

**# Direction bits on binary I/O**

**# VAL 0=input, 1=output**

**file "$(MEASCOMP)/measCompApp/Db/measCompBinaryDir.template"**

**{**

**pattern**

**{ P, R, MASK, VAL, PORT, ADDR}**

**{ 1608G:, Bd1, 0x01 0, 1608G\_1, 0}**

**{ 1608G:, Bd2, 0x02 0, 1608G\_1, 0}**

**{ 1608G:, Bd3, 0x04 0, 1608G\_1, 0}**

**{ 1608G:, Bd4, 0x08 0, 1608G\_1, 0}**

**{ 1608G:, Bd5, 0x10 1, 1608G\_1, 0}**

**{ 1608G:, Bd6, 0x20 1, 1608G\_1, 0}**

**{ 1608G:, Bd7, 0x40 1, 1608G\_1, 0}**

**{ 1608G:, Bd8, 0x80 1, 1608G\_1, 0}**

**}**

Measurement Computing 1608GX-2A0 Driver

Version 4

* Add digital inputs
	+ 1 new parameter, DIGITAL\_INPUT
* Add a poller thread to read digital inputs; their records are SCAN=I/O Intr
* 314 lines of code (~60 more than Version 3)

/\* drvUSB1608G\_V2.cpp

\* This version implements digital **inputs and** outputs, simple analog inputs and simple analog outputs, with a poller thread

 \*

 \* Mark Rivers

 \* April 14, 2012

\*/

#include <iocsh.h>

#include <epicsExport.h>

#include <epicsThread.h>

#include <asynPortDriver.h>

#include "cbw.h"

static const char \*driverName = "USB1608G";

// Analog output parameters

#define analogOutValueString "ANALOG\_OUT\_VALUE"

// Analog input parameters

#define analogInValueString "ANALOG\_IN\_VALUE"

#define analogInRangeString "ANALOG\_IN\_RANGE"

// Digital I/O parameters

#define digitalDirectionString "DIGITAL\_DIRECTION"

**#define digitalInputString "DIGITAL\_INPUT"**

#define digitalOutputString "DIGITAL\_OUTPUT"

#define NUM\_ANALOG\_IN 16 // Number analog inputs on 1608G

#define NUM\_ANALOG\_OUT 2 // Number of analog outputs on 1608G

#define NUM\_IO\_BITS 8 // Number of digital I/O bits on 1608G

#define MAX\_SIGNALS NUM\_ANALOG\_IN

**#define DEFAULT\_POLL\_TIME 0.01**

/\*\* Class definition for the USB1608G class

 \*/

class USB1608G : public asynPortDriver {

public:

 USB1608G(const char \*portName, int boardNum);

 /\* These are the methods that we override from asynPortDriver \*/

 virtual asynStatus writeInt32(asynUser \*pasynUser, epicsInt32 value);

 virtual asynStatus readInt32(asynUser \*pasynUser, epicsInt32 \*value);

virtual asynStatus getBounds(asynUser \*pasynUser, epicsInt32 \*low,

 epicsInt32 \*high);

virtual asynStatus writeUInt32Digital(asynUser \*pasynUser, epicsUInt32 value,

 epicsUInt32 mask);

 virtual void report(FILE \*fp, int details);

 **// These should be private but are called from C**

 **virtual void pollerThread(void);**

protected:

 // Analog output parameters

 int analogOutValue\_;

 #define FIRST\_USB1608G\_PARAM analogOutValue\_

 // Analog input parameters

 int analogInValue\_;

 int analogInRange\_;

 // Digital I/O parameters

 int digitalDirection\_;

 **int digitalInput\_;**

 int digitalOutput\_;

 #define LAST\_USB1608G\_PARAM digitalOutput\_

private:

 int boardNum\_;

 **double pollTime\_;**

 **int forceCallback\_;**

};

#define NUM\_PARAMS (&LAST\_USB1608G\_PARAM - &FIRST\_USB1608G\_PARAM + 1)

**static void pollerThreadC(void \* pPvt)**

**{**

 **USB1608G \*pUSB1608G = (USB1608G \*)pPvt;**

 **pUSB1608G->pollerThread();**

**}**

/\*\* Constructor for the USB1608G class \*/

USB1608G::USB1608G(const char \*portName, int boardNum)

 : asynPortDriver(portName, MAX\_SIGNALS, NUM\_PARAMS,

 // Interfaces that we implement

 asynInt32Mask | asynUInt32DigitalMask | asynDrvUserMask,

 // Interfaces that do callbacks

 asynUInt32DigitalMask,

 ASYN\_MULTIDEVICE | ASYN\_CANBLOCK, 1,

 /\* ASYN\_CANBLOCK=1, ASYN\_MULTIDEVICE=1, autoConnect=1 \*/

 0, 0), /\* Default priority and stack size \*/

 boardNum\_(boardNum),

 **pollTime\_(DEFAULT\_POLL\_TIME),**

 **forceCallback\_(1)**

{

 // Analog output parameters

 createParam(analogOutValueString, asynParamInt32, &analogOutValue\_);

 // Analog input parameters

 createParam(analogInValueString, asynParamInt32, &analogInValue\_);

 createParam(analogInRangeString, asynParamInt32, &analogInRange\_);

 // Digital I/O parameters

 createParam(digitalDirectionString, asynParamUInt32Digital, &digitalDirection\_);

 **createParam(digitalInputString, asynParamUInt32Digital, &digitalInput\_);**

 createParam(digitalOutputString, asynParamUInt32Digital, &digitalOutput\_);

 **/\* Start the thread to poll digital inputs and do callbacks to device support \*/**

 **epicsThreadCreate("USB1608GPoller",**

 **epicsThreadPriorityLow,**

 **epicsThreadGetStackSize(epicsThreadStackMedium),**

 **(EPICSTHREADFUNC)pollerThreadC,**

 **this);**

}

**void USB1608G::pollerThread()**

**{**

 **/\* This function runs in a separate thread. It waits for the poll time \*/**

 **static const char \*functionName = "pollerThread";**

 **epicsUInt32 newValue, changedBits, prevInput=0;**

 **unsigned short biVal;;**

 **int i, status;**

 **while(1) {**

 **lock();**

 **// Read the digital inputs**

 **status = cbDIn(boardNum\_, AUXPORT, &biVal);**

 **if (status)**

 **asynPrint(pasynUserSelf, ASYN\_TRACE\_ERROR,**

 **"%s:%s: ERROR calling cbDIn, status=%d\n",**

 **driverName, functionName, status);**

 **newValue = biVal;**

 **changedBits = newValue ^ prevInput;**

 **if (forceCallback\_ || (changedBits != 0)) {**

 **prevInput = newValue;**

 **forceCallback\_ = 0;**

 **setUIntDigitalParam(digitalInput\_, newValue, 0xFFFFFFFF);**

 **}**

 **for (i=0; i<MAX\_SIGNALS; i++) {**

 **callParamCallbacks(i);**

 **}**

 **unlock();**

 **epicsThreadSleep(pollTime\_);**

 **}**

**}**

**measCompBinaryIn.template**

**record(bi, "$(P)$(R)")**

**{**

 **field(DTYP, "asynUInt32Digital")**

 **field(INP, "@asynMask($(PORT),$(ADDR),$(MASK))DIGITAL\_INPUT")**

 **field(ZNAM, "Low")**

 **field(ONAM, "High")**

 **field(SCAN, "I/O Intr")**

**}**

**measCompLongIn.template**

**record(longin, "$(P)$(R)")**

**{**

 **field(DTYP, "asynUInt32Digital")**

 **field(INP, "@asynMask($(PORT),$(ADDR),$(MASK))DIGITAL\_INPUT")**

 **field(SCAN, "I/O Intr")**

**}**

**1608G.substitutions\_V4**

**file "$(MEASCOMP)/measCompApp/Db/measCompLongIn.template"**

**{**

**pattern**

**{ P, R, MASK, PORT, ADDR}**

**{ 1608G:, Li, 0xFF, 1608G\_1, 0}**

**}**

**file "$(MEASCOMP)/measCompApp/Db/measCompBinaryIn.template"**

**{**

**pattern**

**{ P, R, MASK, PORT, ADDR}**

**{ 1608G:, Bi1, 0x01 1608G\_1, 0}**

**{ 1608G:, Bi2, 0x02 1608G\_1, 0}**

**{ 1608G:, Bi3, 0x04 1608G\_1, 0}**

**{ 1608G:, Bi4, 0x08 1608G\_1, 0}**

**{ 1608G:, Bi5, 0x10 1608G\_1, 0}**

**{ 1608G:, Bi6, 0x20 1608G\_1, 0}**

**{ 1608G:, Bi7, 0x40 1608G\_1, 0}**

**{ 1608G:, Bi8, 0x80 1608G\_1, 0}**

**}**

Measurement Computing 1608GX-2A0 Driver

Version 5

* Add pulse generator output and counter inputs
	+ 8 new parameters, PULSE\_RUN, PULSE\_PERIOD, PULSE\_WIDTH, PULSE\_DELAY, PULSE\_COUNT, PULSE\_IDLE\_STATE, COUNTER\_VALUE, COUNTER\_RESET
* Counter inputs are polled in poller thread, SCAN=I/O Intr
* 484 lines of code (~170 more than Version 4)

/\* drvUSB1608G\_V5.cpp

 \*

 \* Driver for Measurement Computing USB-1608G multi-function DAQ board using asynPortDriver base class

 \*

 \* This version implements digital inputs and outputs, simple analog inputs and simple analog outputs, will a poller thread

 \*

 \* Mark Rivers

 \* April 14, 2012

\*/

#include <iocsh.h>

#include <epicsExport.h>

#include <epicsThread.h>

#include <asynPortDriver.h>

#include "cbw.h"

static const char \*driverName = "USB1608G";

// Analog output parameters

#define analogOutValueString "ANALOG\_OUT\_VALUE"

// Analog input parameters

#define analogInValueString "ANALOG\_IN\_VALUE"

#define analogInRangeString "ANALOG\_IN\_RANGE"

// Digital I/O parameters

#define digitalDirectionString "DIGITAL\_DIRECTION"

#define digitalInputString "DIGITAL\_INPUT"

#define digitalOutputString "DIGITAL\_OUTPUT"

**// Pulse output parameters**

**#define pulseGenRunString "PULSE\_RUN"**

**#define pulseGenPeriodString "PULSE\_PERIOD"**

**#define pulseGenWidthString "PULSE\_WIDTH"**

**#define pulseGenDelayString "PULSE\_DELAY"**

**#define pulseGenCountString "PULSE\_COUNT"**

**#define pulseGenIdleStateString "PULSE\_IDLE\_STATE"**

**// Counter parameters**

**#define counterCountsString "COUNTER\_VALUE"**

**#define counterResetString "COUNTER\_RESET"**

**#define MIN\_FREQUENCY 0.0149**

**#define MAX\_FREQUENCY 32e6**

**#define MIN\_DELAY 0.**

**#define MAX\_DELAY 67.11**

#define NUM\_ANALOG\_IN 16 // Number analog inputs on 1608G

#define NUM\_ANALOG\_OUT 2 // Number of analog outputs on 1608G

**#define NUM\_COUNTERS 2 // Number of counters on 1608G**

#define NUM\_TIMERS 1 // Number of timers on 1608G

#define NUM\_IO\_BITS 8 // Number of digital I/O bits on 1608G

#define MAX\_SIGNALS NUM\_ANALOG\_IN

#define DEFAULT\_POLL\_TIME 0.01

/\*\* Class definition for the USB1608G class

 \*/

class USB1608G : public asynPortDriver {

public:

 USB1608G(const char \*portName, int boardNum);

 /\* These are the methods that we override from asynPortDriver \*/

 virtual asynStatus writeInt32(asynUser \*pasynUser, epicsInt32 value);

 virtual asynStatus readInt32(asynUser \*pasynUser, epicsInt32 \*value);

 virtual asynStatus writeFloat64(asynUser \*pasynUser, epicsFloat64 value);

virtual asynStatus getBounds(asynUser \*pasynUser, epicsInt32 \*low,

 epicsInt32 \*high);

virtual asynStatus writeUInt32Digital(asynUser \*pasynUser, epicsUInt32 value,

 epicsUInt32 mask);

 virtual void report(FILE \*fp, int details);

 // These should be private but are called from C

 virtual void pollerThread(void);

protected:

 **// Pulse generator parameters**

 **int pulseGenRun\_;**

 **#define FIRST\_USB1608G\_PARAM pulseGenRun\_**

 **int pulseGenPeriod\_;**

 **int pulseGenWidth\_;**

 **int pulseGenDelay\_;**

 **int pulseGenCount\_;**

 **int pulseGenIdleState\_;**

 **// Counter parameters**

 **int counterCounts\_;**

 **int counterReset\_;**

 // Analog output parameters

 int analogOutValue\_;

 // Analog input parameters

 int analogInValue\_;

 int analogInRange\_;

 // Digital I/O parameters

 int digitalDirection\_;

 int digitalInput\_;

 int digitalOutput\_;

 #define LAST\_USB1608G\_PARAM digitalOutput\_

private:

 int boardNum\_;

 double pollTime\_;

 int forceCallback\_;

 **int startPulseGenerator();**

 **int stopPulseGenerator();**

 **int pulseGenRunning\_;**

};

#define NUM\_PARAMS (&LAST\_USB1608G\_PARAM - &FIRST\_USB1608G\_PARAM + 1)

/\*\* Constructor for the USB1608G class

 \*/

USB1608G::USB1608G(const char \*portName, int boardNum)

 : asynPortDriver(portName, MAX\_SIGNALS, NUM\_PARAMS,

 // Interfaces that we implement

 asynInt32Mask | **asynFloat64Mask** | asynUInt32DigitalMask | asynDrvUserMask,

 // Interfaces that do callbacks

 **asynInt32Mask | asynFloat64Mask** | asynUInt32DigitalMask,

 /\* ASYN\_CANBLOCK=1, ASYN\_MULTIDEVICE=1, autoConnect=1 \*/

 ASYN\_MULTIDEVICE | ASYN\_CANBLOCK, 1, 0, 0), /\* Default priority and stack size \*/

 boardNum\_(boardNum),

 pollTime\_(DEFAULT\_POLL\_TIME),

 forceCallback\_(1)

{

 **// Pulse generator parameters**

 **createParam(pulseGenRunString, asynParamInt32, &pulseGenRun\_);**

 **createParam(pulseGenPeriodString, asynParamFloat64, &pulseGenPeriod\_);**

 **createParam(pulseGenWidthString, asynParamFloat64, &pulseGenWidth\_);**

 **createParam(pulseGenDelayString, asynParamFloat64, &pulseGenDelay\_);**

 **createParam(pulseGenCountString, asynParamInt32, &pulseGenCount\_);**

 **createParam(pulseGenIdleStateString, asynParamInt32, &pulseGenIdleState\_);**

 **// Counter parameters**

 **createParam(counterCountsString, asynParamInt32, &counterCounts\_);**

 **createParam(counterResetString, asynParamInt32, &counterReset\_);**

 // Analog output parameters

 createParam(analogOutValueString, asynParamInt32, &analogOutValue\_);

 // Analog input parameters

 createParam(analogInValueString, asynParamInt32, &analogInValue\_);

 createParam(analogInRangeString, asynParamInt32, &analogInRange\_);

 // Digital I/O parameters

 createParam(digitalDirectionString, asynParamUInt32Digital, &digitalDirection\_);

 createParam(digitalInputString, asynParamUInt32Digital, &digitalInput\_);

 createParam(digitalOutputString, asynParamUInt32Digital, &digitalOutput\_);

 /\* Start the thread to poll digital inputs and do callbacks to

 \* device support \*/

 epicsThreadCreate("USB1608GPoller",

 epicsThreadPriorityLow,

 epicsThreadGetStackSize(epicsThreadStackMedium),

 (EPICSTHREADFUNC)pollerThreadC,

 this);

}

**int USB1608G::startPulseGenerator()**

**{**

 **int status=0;**

 **double frequency, period, width, delay;**

 **int timerNum=0;**

 **double dutyCycle;**

 **int count, idleState;**

 **static const char \*functionName = "startPulseGenerator";**

 **getDoubleParam (timerNum, pulseGenPeriod\_, &period);**

 **getDoubleParam (timerNum, pulseGenWidth\_, &width);**

 **getDoubleParam (timerNum, pulseGenDelay\_, &delay);**

 **getIntegerParam(timerNum, pulseGenCount\_, &count);**

 **getIntegerParam(timerNum, pulseGenIdleState\_, &idleState);**

 **frequency = 1./period;**

 **if (frequency < MIN\_FREQUENCY) frequency = MIN\_FREQUENCY;**

 **if (frequency > MAX\_FREQUENCY) frequency = MAX\_FREQUENCY;**

 **dutyCycle = width \* frequency;**

 **period = 1. / frequency;**

 **if (dutyCycle <= 0.) dutyCycle = .0001;**

 **if (dutyCycle >= 1.) dutyCycle = .9999;**

 **if (delay < MIN\_DELAY) delay = MIN\_DELAY;**

 **if (delay > MAX\_DELAY) delay = MAX\_DELAY;**

**status = cbPulseOutStart(boardNum\_, timerNum, &frequency, &dutyCycle, count,**

 **&delay, idleState, 0);**

 **if (status != 0) {**

 **asynPrint(pasynUserSelf, ASYN\_TRACE\_ERROR,**

 **"%s:%s: started pulse generator %d period=%f, width=%f, count=%d,**

 **delay=%f, idleState=%d, status=%d\n",**

 **driverName, functionName, timerNum, period, width, count, delay,**

 **idleState, status);**

 **return status;**

 **}**

**// We may not have gotten the frequency, dutyCycle, and delay we asked for,**

**// set the actual values in the parameter library**

 **pulseGenRunning\_ = 1;**

 **period = 1. / frequency;**

 **width = period \* dutyCycle;**

 **asynPrint(pasynUserSelf, ASYN\_TRACE\_FLOW,**

 **"%s:%s: started pulse generator %d actual frequency=%f, actual period=%f,**

 **actual width=%f, actual delay=%f\n",**

 **driverName, functionName, timerNum, frequency, period, width, delay);**

 **setDoubleParam(timerNum, pulseGenPeriod\_, period);**

 **setDoubleParam(timerNum, pulseGenWidth\_, width);**

 **setDoubleParam(timerNum, pulseGenDelay\_, delay);**

 **return 0;**

**}**

**int USB1608G::stopPulseGenerator()**

**{**

 **pulseGenRunning\_ = 0;**

 **return cbPulseOutStop(boardNum\_, 0);**

**}**

asynStatus USB1608G::writeInt32(asynUser \*pasynUser, epicsInt32 value)

{

 int addr;

 int function = pasynUser->reason;

 int status=0;

 static const char \*functionName = "writeInt32";

 this->getAddress(pasynUser, &addr);

 setIntegerParam(addr, function, value);

 **// Pulse generator functions**

 **if (function == pulseGenRun\_) {**

 **// Allow starting a run even if it thinks its running,**

 **// since there is no way to know when it got done if Count!=0**

 **if (value) {**

 **status = startPulseGenerator();**

 **}**

 **else if (!value && pulseGenRunning\_) {**

 **status = stopPulseGenerator();**

 **}**

 **}**

 **if ((function == pulseGenCount\_) ||**

 **(function == pulseGenIdleState\_)) {**

 **if (pulseGenRunning\_) {**

 **status = stopPulseGenerator();**

 **status |= startPulseGenerator();**

 **}**

 **}**

 **// Counter functions**

 **if (function == counterReset\_) {**

 **// LOADREG0=0, LOADREG1=1, so we use addr**

 **status = cbCLoad32(boardNum\_, addr, 0);**

 **}**

 // Analog output functions

 if (function == analogOutValue\_) {

 status = cbAOut(boardNum\_, addr, BIP10VOLTS, value);

 }

 callParamCallbacks(addr);

 if (status == 0) {

 asynPrint(pasynUser, ASYN\_TRACEIO\_DRIVER,

 "%s:%s, port %s, wrote %d to address %d\n",

 driverName, functionName, this->portName, value, addr);

 } else {

 asynPrint(pasynUser, ASYN\_TRACE\_ERROR,

 "%s:%s, port %s, ERROR writing %d to address %d, status=%d\n",

 driverName, functionName, this->portName, value, addr, status);

 }

 return (status==0) ? asynSuccess : asynError;

}

**asynStatus USB1608G::writeFloat64(asynUser \*pasynUser, epicsFloat64 value)**

**{**

 **int addr;**

 **int function = pasynUser->reason;**

 **int status=0;**

 **static const char \*functionName = "writeFloat64";**

 **this->getAddress(pasynUser, &addr);**

 **setDoubleParam(addr, function, value);**

 **// Pulse generator functions**

 **if ((function == pulseGenPeriod\_) ||**

 **(function == pulseGenWidth\_) ||**

 **(function == pulseGenDelay\_)) {**

 **if (pulseGenRunning\_) {**

 **status = stopPulseGenerator();**

 **status |= startPulseGenerator();**

 **}**

 **}**

 **callParamCallbacks(addr);**

 **if (status == 0) {**

 **asynPrint(pasynUser, ASYN\_TRACEIO\_DRIVER,**

 **"%s:%s, port %s, wrote %d to address %d\n",**

 **driverName, functionName, this->portName, value, addr);**

 **} else {**

 **asynPrint(pasynUser, ASYN\_TRACE\_ERROR,**

 **"%s:%s, port %s, ERROR writing %f to address %d, status=%d\n",**

 **driverName, functionName, this->portName, value, addr, status);**

 **}**

 **return (status==0) ? asynSuccess : asynError;**

**}**

void USB1608G::pollerThread()

{

 /\* This function runs in a separate thread. It waits for the poll time \*/

 static const char \*functionName = "pollerThread";

 epicsUInt32 newValue, changedBits, prevInput=0;

 unsigned short biVal;;

 **unsigned long countVal;**

 int i;

 int status;

 while(1) {

 lock();

 **// Read the counter inputs**

 **for (i=0; i<NUM\_COUNTERS; i++) {**

 **status = cbCIn32(boardNum\_, i, &countVal);**

 **if (status)**

 **asynPrint(pasynUserSelf, ASYN\_TRACE\_ERROR,**

 **"%s:%s: ERROR calling cbCIn32, status=%d\n",**

 **driverName, functionName, status);**

 **setIntegerParam(i, counterCounts\_, countVal);**

 **}**

 // Read the digital inputs

 status = cbDIn(boardNum\_, AUXPORT, &biVal);

 if (status)

 asynPrint(pasynUserSelf, ASYN\_TRACE\_ERROR,

 "%s:%s: ERROR calling cbDIn, status=%d\n",

 driverName, functionName, status);

 newValue = biVal;

 changedBits = newValue ^ prevInput;

 if (forceCallback\_ || (changedBits != 0)) {

 prevInput = newValue;

 forceCallback\_ = 0;

 setUIntDigitalParam(digitalInput\_, newValue, 0xFFFFFFFF);

 }

 for (i=0; i<MAX\_SIGNALS; i++) {

 callParamCallbacks(i);

 }

 unlock();

 epicsThreadSleep(pollTime\_);

 }

}

**measCompPulseGen.template**

**###################################################################**

**# Pulse start/stop #**

**###################################################################**

**record(bo, "$(P)$(R)Run")**

**{**

 **field(DTYP, "asynInt32")**

 **field(OUT, "@asyn($(PORT),$(ADDR))PULSE\_RUN")**

 **field(ZNAM, "Stop")**

 **field(ZSV, "NO\_ALARM")**

 **field(ONAM, "Run")**

 **field(OSV, "MINOR")**

**}**

**# NOTE: The records for the period and the frequency are a bit**

**# complex because we want to be able to change either ao record**

**# and have the other one update**

**###################################################################**

**# Pulse period #**

**###################################################################**

**record(ao, "$(P)$(R)Period")**

**{**

 **field(PINI, "YES")**

 **field(DTYP, "asynFloat64")**

 **field(OUT, "@asyn($(PORT),$(ADDR))PULSE\_PERIOD")**

 **field(VAL, "0.001")**

 **field(PREC, "$(PREC)")**

 **field(FLNK, "$(P)$(R)CalcFrequency")**

**}**

**###################################################################**

**# Calculate frequency based on new period #**

**###################################################################**

**record(calcout, "$(P)$(R)CalcFrequency")**

**{**

 **field(INPA, "$(P)$(R)Period")**

 **field(CALC, "1/A")**

 **field(SDIS, "$(P)$(R)Frequency.PROC")**

 **field(DISV, "1")**

 **field(OUT, "$(P)$(R)Frequency PP MS")**

**}**

**###################################################################**

**# Pulse frequency #**

**###################################################################**

**record(ao, "$(P)$(R)Frequency")**

**{**

 **field(PREC, "$(PREC)")**

 **field(FLNK, "$(P)$(R)CalcPeriod PP MS")**

**}**

**###################################################################**

**# Calculate period based on new frequency #**

**###################################################################**

**record(calcout, "$(P)$(R)CalcPeriod")**

**{**

 **field(INPA, "$(P)$(R)Frequency")**

 **field(CALC, "1/A")**

 **field(SDIS, "$(P)$(R)Period.PROC")**

 **field(DISV, "1")**

 **field(OUT, "$(P)$(R)Period PP MS")**

**}**

**###################################################################**

**# Pulse width #**

**###################################################################**

**record(ao, "$(P)$(R)Width")**

**{**

 **field(PINI, "YES")**

 **field(DTYP, "asynFloat64")**

 **field(OUT, "@asyn($(PORT),$(ADDR))PULSE\_WIDTH")**

 **field(VAL, "0.0001")**

 **field(PREC, "$(PREC)")**

**}**

**###################################################################**

**# Pulse delay #**

**###################################################################**

**record(ao, "$(P)$(R)Delay")**

**{**

 **field(PINI, "YES")**

 **field(DTYP, "asynFloat64")**

 **field(OUT, "@asyn($(PORT),$(ADDR))PULSE\_DELAY")**

 **field(VAL, "0.")**

 **field(PREC, "$(PREC)")**

**}**

**###################################################################**

**# Pulse count #**

**###################################################################**

**record(longout, "$(P)$(R)Count")**

**{**

 **field(PINI, "YES")**

 **field(DTYP, "asynInt32")**

 **field(OUT, "@asyn($(PORT),$(ADDR))PULSE\_COUNT")**

**}**

**###################################################################**

**# Pulse idle state #**

**###################################################################**

**record(bo, "$(P)$(R)IdleState")**

**{**

 **field(PINI, "YES")**

 **field(DTYP, "asynInt32")**

 **field(OUT, "@asyn($(PORT),$(ADDR))PULSE\_IDLE\_STATE")**

 **field(ZNAM, "Low")**

 **field(ONAM, "High")**

 **field(VAL, "0")**

**}**

**1608GCounter.template**

**record(longin, "$(P)$(R)Counts")**

**{**

 **field(DTYP, "asynInt32")**

 **field(INP, "@asyn($(PORT),$(ADDR))COUNTER\_VALUE")**

 **field(SCAN, "I/O Intr")**

**}**

**record(bo, "$(P)$(R)Reset")**

**{**

 **field(PINI, "YES")**

 **field(DTYP, "asynInt32")**

 **field(OUT, "@asyn($(PORT),$(ADDR))COUNTER\_RESET")**

 **field(VAL, "1")**

**}**

**1608G.substitutions\_V5**

**file "$(MEASCOMP)/measCompApp/Db/measCompPulseGen.template"**

**{**

**pattern**

**{ P, R, PORT, ADDR, PREC}**

**{ 1608G:, PulseGen1, 1608G\_1, 0, 4}**

**}**

**file "$(MEASCOMP)/measCompApp/Db/1608GCounter.template"**

**{**

**pattern**

**{ P, R, PORT, ADDR}**

**{ 1608G:, Counter1, 1608G\_1, 0}**

**{ 1608G:, Counter2, 1608G\_1, 1}**

**}**

Measurement Computing 1608GX-2A0 Driver

Full Released Version

* Add waveform generator with both predefined and user-defined waveforms
* Add 8-channel waveform digitizer
* Add trigger support
* 42 new parameters
* 1254 lines of code (~770 more than Version 4)
* Supports virtually all features of 1608GX-2A0
* New version written in 2016 has been renamed to drvMultiFunction and supports other Measurement Computing Multifunction devices as well.

