// MarkerLibTest.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <vector>

#include "commondefs.h"

#include "LoopingSegment.h"

#include "lec\_public\_exports.h"

const wchar\_t \*wsIPAddress=L"127.0.0.1";

//const wchar\_t \*wcsDefaultDeviceID=L"00:50:c2:c8:c1:3e";

//const wchar\_t \*wcsDefaultDeviceID=L"00:50:c2:c8:c0:00";

const wchar\_t \*wcsDefaultLANDeviceID=L"00:50:c2:4f:a2:f9";

const wchar\_t \*wcsDefaultPCIDeviceID=L"\\?\pci#ven\_1597&dev\_0301&subsys\_00011597&rev\_00#0000000101000a3500#{b74cfec2-9366-454a-ba71-7c27b51470a4}";

wchar\_t wcsDeviceID[100];

int seqNum = 0;

using namespace std;

const int MAC\_ADDR\_LEN=17;

const int PCI\_ID\_LEN=106;

void \_\_stdcall DeviceDetectionCallback(LanmarkControls::HardwareModel AModel, unsigned char\* ADeviceBytes, int AByteSize)

{

}

void \_\_stdcall JobEventCallback (unsigned \_\_int64 AJobRef,

unsigned \_\_int64 AObjRef,

LanmarkControls::LECJobEvents AEventCode,

LanmarkControls::LECResponse AResponseCode,

int APayload2,

int APayload3,

int APayload4)

{

}

void \_\_stdcall ActionEventCallback (unsigned \_\_int64 AJobRef,

LanmarkControls::UndoRedoState AUndo,

LanmarkControls::UndoRedoState ARedo)

{

}

void \_\_stdcall DeviceEventCallback(wchar\_t\* ADeviceID, unsigned int AHighPayload, unsigned int ALowPayload)

{

}

/\*

int test1(vector<LECBaseDevice \*> \*pDevices)

{

int nRet=0;

//Load Data

char \*str1="Lanmark Controls Inc";

char \*str2="Wenlong";

char \*str3="Data";

char \*str4="Test";

LoadSegment(0,(unsigned char\*)str1,strlen(str1));

LoadSegment(2,(unsigned char\*)str2,strlen(str2));

LoadSegment(3,(unsigned char\*)str3,strlen(str3));

LoadSegment(4,(unsigned char\*)str4,strlen(str4));

//Set the connection details

CONNECTDETAILS details;

details.devicemodel=LanmarkControls::LEC2\_USB;

details.clienttype=LanmarkControls::ds\_DeviceUSB;

//use the first available device

wcscpy\_s(details.deviceid,DEVICE\_ID\_LENGTH,(\*(pDevices->begin()))->FDeviceID);

printf("Execute segments\n");

nRet=ExecuteSegment(details,0);

if (nRet==LanmarkControls::Success)

printf("Succeeded in sending data to the driver\n");

else

printf("Failed to send data to the driver, Error Code:%d\n",nRet);

getchar();

//test looping

//Create a list of looping segments

LoopingSegment \*p1=new LoopingSegment();

p1->SetIndex(0);

//Then we loop on "Control Inc"

p1->SetLoopPoint(8);

LoopingSegment \*p2=new LoopingSegment();

p2->SetIndex(2);

//Then we loop on "long"

p2->SetLoopPoint(3);

LoopingSegment \*p3=new LoopingSegment();

p3->SetIndex(3);

//Then we loop on "Liu"

p3->SetLoopPoint(0);

LoopingSegment \*p4=new LoopingSegment();

p4->SetIndex(4);

//Then we loop on "Jason"

p4->SetLoopPoint(0);

//Link those segments

p1->SetNext(p2);

p2->SetNext(p3);

p3->SetNext(p4);

p4->SetNext(NULL);

printf("\*\*\*\*\*\*\*\*\*Looping\*\*\*\*\*\*\*\*\*\n");

LoopSegment(details,p1);

//sleep and abort

::Sleep(1000);

AbortList(details);

//wait for abort

::Sleep(1000);

return nRet;

}\*/

//This is a really meaningful sample

/\*

int test2(vector<wchar\_t\*> \*pDeviceIDList, MarkerLibraryRT\* pMarkerLibrary)

{

int nRet=0;

//Load Data

char \*str1="Lanmark Controls Inc";

char \*str2="Long String!Long String!Long String!Long String!Long String!Long String!Long String!";

char \*str3="Data";

char \*str4="Test";

//pMarkerLibrary.LoadSegment(0,(unsigned char\*)str1,strlen(str1));

//pMarkerLibrary.LoadSegment(2,(unsigned char\*)str2,strlen(str2));

//pMarkerLibrary.LoadSegment(3,(unsigned char\*)str3,strlen(str3));

//pMarkerLibrary.LoadSegment(4,(unsigned char\*)str4,strlen(str4));

//Set the connection details

CONNECTDETAILS details;

details.devicemodel=LanmarkControls::LEC2\_USB;

details.clienttype=LanmarkControls::ds\_DeviceUSB;

//use the first available device

wcscpy\_s(details.deviceid,DEVICE\_ID\_LENGTH,(\*(pDeviceIDList->begin())));

printf("Execute segments:%ws\n",details.deviceid);

nRet=ExecuteList(details.deviceid,0);

if (nRet==LanmarkControls::Success)

printf("Succeeded in sending data to the driver\n");

else

printf("Failed to send data to the driver, Error Code:%d\n",nRet);

getchar();

//test looping

//Create a list of looping segments

LoopingSegment \*p1=new LoopingSegment();

p1->SetIndex(0);

//Then we loop on "Control Inc"

p1->SetLoopPoint(8);

LoopingSegment \*p2=new LoopingSegment();

p2->SetIndex(2);

//Then we loop on "long"

p2->SetLoopPoint(8);

LoopingSegment \*p3=new LoopingSegment();

p3->SetIndex(3);

//Then we loop on "Data"

p3->SetLoopPoint(0);

LoopingSegment \*p4=new LoopingSegment();

p4->SetIndex(4);

//Then we loop on "Test"

p4->SetLoopPoint(0);

//Link those segments

p1->SetNext(p2);

p2->SetNext(p3);

p3->SetNext(p4);

p4->SetNext(p2);

printf("\*\*\*\*\*\*\*\*\*Looping\*\*\*\*\*\*\*\*\*\n");

//LoopSegment(details,p1);

nRet=LoopList(details.deviceid, p1);

//sleep and abort

::Sleep(1000);

//AbortList(details);

//wait for abort

::Sleep(1000);

return nRet;

return 0;

}

\*/

int stepSize=10;

int stepPeriod=300;

int indexZero=0;

int indexOne=1;

int indexTwo=2;

int indexThree=3;

int indexFour=4;

int indexFive=5;

int length=1000;

int delay=1;

//This tests LEC1 LAN device

int test3(vector<wchar\_t\*> \*pDeviceIDList)

{

int nRet=0;

for (vector<wchar\_t\*>::iterator iter=pDeviceIDList->begin();

iter!=pDeviceIDList->end();++iter)

{

printf("%ws\n",(\*iter));

//wchar\_t \*deviceID=new wchar\_t[50];

//wcscpy\_s(deviceID,50,wcsDeviceID);

//Segment 1:

// Square:5000\*5000

Set\_Jump\_Speed(0,stepPeriod,stepSize);

Set\_Mark\_Speed(0,stepPeriod,stepSize);

Jump\_Abs(0,-5000,-5000,100,length,delay);

Mark\_Abs(0,-5000,5000,100,length,delay);

Mark\_Abs(0,5000,5000,100,length,delay);

Mark\_Abs(0,5000,-5000,100,length,delay);

Mark\_Abs(0,-5000,-5000,100,length,delay);

//Segment 2:

// Square:7500\*7500

Set\_Jump\_Speed(indexOne,stepPeriod,stepSize);

Set\_Mark\_Speed(indexOne,stepPeriod,stepSize);

Jump\_Abs(indexOne,-7500,-7500,100,length,delay);

Mark\_Abs(indexOne,-7500,7500,100,length,delay);

Mark\_Abs(indexOne,7500,7500,100,length,delay);

Mark\_Abs(indexOne,7500,-7500,100,length,delay);

Mark\_Abs(indexOne,-7500,-7500,100,length,delay);

//Segment 3:

//Square:10000\*10000

Set\_Jump\_Speed(indexTwo,stepPeriod,stepSize);

Set\_Mark\_Speed(indexTwo,stepPeriod,stepSize);

Jump\_Abs(indexTwo,-10000,-10000,100,length,delay);

Mark\_Abs(indexTwo,-10000,10000,100,length,delay);

Mark\_Abs(indexTwo,10000,10000,100,length,delay);

Mark\_Abs(indexTwo,10000,-10000,100,length,delay);

Mark\_Abs(indexTwo,-10000,-10000,100,length,delay);

//Segment 4:

//Triangle

Set\_Jump\_Speed(indexThree,stepPeriod,stepSize);

Set\_Mark\_Speed(indexThree,stepPeriod,stepSize);

Jump\_Abs(indexThree,0,5000,100,length,delay);

Mark\_Abs(indexThree,5000,0,100,length,delay);

Mark\_Abs(indexThree,-5000,0,100,length,delay);

//Segment 5:

//Triangle

Set\_Jump\_Speed(indexFour,stepPeriod,stepSize);

Set\_Mark\_Speed(indexFour,stepPeriod,stepSize);

Jump\_Abs(indexFour,0,20000,100,length,delay);

Mark\_Abs(indexFour,20000,0,100,length,delay);

Mark\_Abs(indexFour,-20000,0,100,length,delay);

printf("Execute segments\n");

Sleep(1000);

int index=0;

while (index <=4)

{

nRet=ExecuteList(\*iter,index);

if (nRet==LanmarkControls::Success)

printf("Succeeded in sending data to the driver\n");

else

printf("Failed to send data to the driver, Error Code:%d\n",nRet);

index++;

Sleep(50);

}

}

Sleep(10000);

//return 0;

//getchar();

//test looping

//Create a list of looping segments

LoopingSegment \*p1=new LoopingSegment();

p1->SetIndex(0);

p1->SetLoopPoint(0);

LoopingSegment \*p2=new LoopingSegment();

p2->SetIndex(1);

p2->SetLoopPoint(0);

LoopingSegment \*p3=new LoopingSegment();

p3->SetIndex(2);

p3->SetLoopPoint(0);

LoopingSegment \*p4=new LoopingSegment();

p4->SetIndex(3);

p4->SetLoopPoint(0);

LoopingSegment \*p5=new LoopingSegment();

p5->SetIndex(4);

p5->SetLoopPoint(0);

//Link those segments

p1->SetNext(p2);

p2->SetNext(p3);

p3->SetNext(p4);

p4->SetNext(p5);

p5->SetNext(p1);

printf("\*\*\*\*\*\*\*\*\*Looping\*\*\*\*\*\*\*\*\*\n");

vector<wchar\_t\*>::iterator iterloop=pDeviceIDList->begin();

Sleep(1000);

nRet=LoopList(\*iterloop, p1);

if (nRet==LanmarkControls::Success)

printf("Succeeded in Looping data\n");

else

printf("Failed to Loop data, Error Code:%d\n",nRet);

//getchar();

vector<wchar\_t\*>::iterator iterAbort = pDeviceIDList->begin();

AbortList(\*iterAbort);

//wait for abort

::Sleep(1000);

//delete [] deviceID;

delete [] \*iterAbort;

return nRet;

}

// This tests the following function:

// -- List Management Functions:

// 1. GetListSize

// -- List Commands:

// 2. Begin\_Job\_Event

// 3. End\_Job\_Event

// 4. Laser\_Signal\_Off

// 5. Laser\_Signal\_On

int test4(vector<wchar\_t\*> \*pDeviceIDList)

{

int nRet = 0;

int lsRet = 0;

unsigned int listSize = 0;

for (vector<wchar\_t\*>::iterator iter = pDeviceIDList->begin();

iter!= pDeviceIDList->end();++iter)

{

printf("%ws\n",(\*iter));

//Segment 1:

//Square:5000\*5000

//OriginalListSize=112

//PackedListSize=144

Begin\_Job\_Event(0);

Laser\_Signal\_On(0);

Set\_Jump\_Speed(0,stepPeriod,stepSize);

Set\_Mark\_Speed(0,stepPeriod,stepSize);

Jump\_Abs(0,-5000,-5000,100,length,delay);

Mark\_Abs(0,-5000,5000,100,length,delay);

Mark\_Abs(0,5000,5000,100,length,delay);

Mark\_Abs(0,5000,-5000,100,length,delay);

Mark\_Abs(0,-5000,-5000,100,length,delay);

Laser\_Signal\_Off(0);

End\_Job\_Event(0);

//Segment 2:

//Triangle

//OriginalListSize=80

//PackedListSize=112

Begin\_Job\_Event(indexOne);

Laser\_Signal\_On(indexOne);

Set\_Jump\_Speed(indexOne,stepPeriod,stepSize);

Set\_Mark\_Speed(indexOne,stepPeriod,stepSize);

Jump\_Abs(indexOne,0,5000,100,length,delay);

Mark\_Abs(indexOne,5000,0,100,length,delay);

Mark\_Abs(indexOne,-5000,0,100,length,delay);

Laser\_Signal\_Off(indexOne);

End\_Job\_Event(indexOne);

printf("Execute segments\n");

Sleep(1000);

DWORD index = 0;

//unsigned int index1 = 0;

unsigned int \*lsPtr = NULL;

lsPtr = &listSize;

while (index <= 1)

{

nRet = ExecuteList(\*iter, index);

//lsRet = GetListSize(index, lsPtr);

//lsRet = MarkerLibraryRT::GetSegmentSize(index, lsPtr);

if (nRet == LanmarkControls::Success)

{

printf("Succeeded in sending data to the driver\n");

/\*if (lsRet = LanmarkControls::Success)

printf("Succeed in getting list size.");

else

printf("Failed in getting list size.");\*/

}

else

printf("Failed to send data to the driver, Error Code:%d\n",nRet);

index++;

//index1++;

Sleep(50);

}

}

vector<wchar\_t\*>::iterator iterAbort = pDeviceIDList->begin();

AbortList(\*iterAbort);

//wait for abort

::Sleep(1000);

int dRet = 0;

DisconnectLECDevice(\*iterAbort);

if (dRet == LanmarkControls::Success)

printf("Succeeded in disconnecting LEC device.\n");

else

printf("Failed in disconnecting LEC device, Error Code:%d\n",dRet);

//delete [] deviceID;

delete [] \*iterAbort;

return nRet;

}

int TestControls(vector<wchar\_t\*>& vecDevIDs)

{

const int EVENT\_NAME\_LEN=100;

printf("Test Controls:\n");

for (vector<wchar\_t\*>::iterator iter=vecDevIDs.begin();

iter!=vecDevIDs.end();++iter)

{

printf("Device ID:%ls\n",\*iter);

//Print out some of the registers

{

//reg MecFPGAReg\_CmdFifoPeek

MecFPGAReg reg(MecFPGAReg\_CmdFifoPeek);

unsigned int val;

if (LanmarkControls::Success==ReadRegister(\*iter,(unsigned int)reg,&val))

printf(" Register MecFPGAReg\_CmdFifoPeek:%x\n",val);

//reg MecFPGAReg\_Status

reg=MecFPGAReg\_Status;

if (LanmarkControls::Success==ReadRegister(\*iter,(unsigned int)reg,&val))

printf(" Register MecFPGAReg\_Status:%x\n",val);

//

}

/\*

//Message Event Name

wchar\_t wcsEventName[EVENT\_NAME\_LEN];

if (LanmarkControls::Success==GetJobMsgName(\*iter,wcsEventName,EVENT\_NAME\_LEN))

printf\_s("\nMessage Event Name:%ws\n",wcsEventName,EVENT\_NAME\_LEN);

//Exception Event Name

if (LanmarkControls::Success==GetExceptionMsgName(\*iter,wcsEventName,EVENT\_NAME\_LEN))

printf\_s("Exception Event Name:%ws\n",wcsEventName,EVENT\_NAME\_LEN);

//Driver status

printf("\n");

\*/

}

return 0;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

LoadMarkerLibrary();

InitializeMarkerLibrary(NULL,NULL,NULL,DeviceDetectionCallback,NULL);

//getchar();

int nRet=InitializeBufferManager(5,200);

vector<wchar\_t\*> vecTemp,vecDevIDs;

bool b=true;

printf("Please choose the device that you want to test:(input the sequence number)\n");

printf("1: LAN\n");

printf("2: PCI\n");

scanf\_s("%d",&seqNum);

printf("Please input the Device ID (mac address or device path)\n");

scanf\_s("%ls",wcsDeviceID,sizeof(wcsDeviceID));

if(seqNum==1){

if (wcslen(wcsDeviceID)!=MAC\_ADDR\_LEN)

{

wcscpy(wcsDeviceID,wcsDefaultLANDeviceID);

printf("Use the default LAN Device ID:%ls\n",wcsDeviceID);

}

}

else if(seqNum==2)

{

if (wcslen(wcsDeviceID)!=PCI\_ID\_LEN)

{

wcscpy(wcsDeviceID,wcsDefaultPCIDeviceID);

printf("Use the default PCI Device path:%ls\n",wcsDeviceID);

}

}

if (wcslen(wcsDeviceID)==0)

//Test all devices. Wait for enumerating all devices.

::Sleep(1000);

while (b)

{

GetLECDeviceIDList(&vecTemp);

for (vector<wchar\_t\*>::iterator iter=vecTemp.begin();

iter!=vecTemp.end();++iter)

{

wcscpy(wcsDeviceID, \*iter);

b=false;

vecDevIDs.push\_back((\*iter));

break;

}

/\*

if (wcslen(wcsDeviceID)==0 //test all device detected by the library

|| wcscmp((\*iter),wcsDeviceID)==0

)

{

//Our device has already been found

vecDevIDs.push\_back((\*iter));

b=false;

}

\*/

}

//GetLECDeviceIDList(&vecDevIDs);

//printf("Get the following devices:\n");

//getchar();

//for (vector<wchar\_t\*>::iterator iter=pDeviceIDList->begin();

//iter!=pDeviceIDList->end();++iter)

//printf("%ws\n",(\*iter));

//if (pDeviceIDList->empty())

//return -1;

//getchar();

//Send some testing data to the first device

//test1(pDevices);

//test2(&vecDevIDs, pMarkerLibrary);

//test3(&vecDevIDs);

test4(&vecDevIDs);

//unsigned int ls=0;

//GetListSize(0,&ls);

//TestControls(vecDevIDs);

vecDevIDs.clear();

for (vector<wchar\_t\*>::iterator iter=vecTemp.begin();

iter!=vecTemp.end();++iter)

delete \*iter;

vecTemp.clear();

ReleaseLibrary();

//getchar();

//getchar();

//exit(0);

printf("End of the program:\n");

//To do : seems some threads are not terminated correctly

return 0;

}