University of BRISTOL

Department Of Physics

FINAL YEAR PROJECT, DISSERTATION OR PHYSICS EDUCATION REPORT

NAME:	Luka Milic
DEGREE COURSE:	Mathematics and Physics (MSci)
PROJECT TITLE:	Entanglement of photons pairs generated in silicon ring resonators
YEAR OF SUBMISSION:	2015
SUPERVISOR:	Damien Bonneau, Josh Silverstone and Mark Thompson
NUMBER OF WORDS:	336 (exclude appendices, references, captions and abstract)

Acknowledgements

Thank you Damien and Josh for your invaluable help throughout my project. Lizzy for working with me in the lab. Imad for encouragement and guidance. Phil for your sarcasm.

Contents

1 Introduction

The endeavour to build a quantum computer holds the promise of solving computational problems which are currently intractable on classical computers. A particularly promising paradigm for this is the linear optical quantum computer (LOQC) model which in theory allows for scalable universal quantum computation. Work on LOQC can be done using bulk optics components but this quickly becomes impractical when the experiments needed to be scaled up to more qubits. Integrated photonics has solved this problem and allowed for experiments with more qubits in a much smaller space. Optical circuits can be implemented on such chips, popular materials are silicon-on-insulator (SOI) , lithium niobate and glass materials. Here we focus on SOI chips as they have many promising properties for the implementation of complex optical circuits.

A key requirement for full implementation of LOQC is a scalable, bright, deterministic and indistinguishable single photon source. Single photon sources in the SOI platform are typically made from the waveguide itself and use the spontaneous four-wave mixing which occurs in silicon due to the third order non-linearity. This report aims to develop a new method of measuring the indistinguishably of the produced photons with a classical technique, exploiting stimulated four-wave mixing. This method collects a Joint Spectrum which is an estimation of the wave function of the produced single photon.

2 Detailed Background and Theory

2.1 Integrated silicon photonics

2.2 On chip four wave mixing

2.3 Ring Resonators

2.4 Bistability

2.5 Self phase modulation

2.6 Schmidt Rank and Purity

3 Method

3.1 Coupling

- Side/Vertical coupling
- Coupling loss
- Blowing up chips
-

3.2 Joint Spectrum

3.3 $g^{(2)}(0)$

4 Results

4.1 Glassgow

4.2 Toshiba

4.3 a-Si

5 Discussion

6 Conclusion

References

[1] Andreas Eckstein, Guillaume Boucher, Aristide Lematre, Pascal Filloux, Ivan Favero, Giuseppe Leo, John E. Sipe, Marco Liscidini, and Sara Ducci. High-resolution spectral characterization of two photon states via classical measurements. *Laser & Photonics Reviews*, 8(5):L76–L80, September 2014.

A Schmidt Number

A.1 Definition

Starting with some abitrary state ψ:

$$|\psi\rangle = \sum_{i,j} \alpha(i,j)|i\rangle_A \otimes |j\rangle_B \tag{1}$$

The schimdt number K of this state measures the degree of entanglement. If $K = 1$ then you can find $|\psi\rangle = |\xi\rangle \otimes |\eta\rangle$ and for $K > 1$ you can find:

$$|\psi\rangle = \sum_{i}^{K} r_i|\xi_i\rangle_A \otimes |\eta_i\rangle_B \tag{2}$$

Note that $1 \leq K \leq D$ where D is the dimension of the system. The purity is the inverse of K so:

$$P = 1/K \tag{3}$$

An expression for K can be found using the density matrix for ψ:

$$\rho_{AB} = |\psi\rangle\langle\psi| = \sum_{i,j,k,l} \alpha(i,j)\alpha^*(k,l)|i\rangle\langle k| \otimes |j\rangle\langle l| \tag{4}$$

$$\rho_A = \mathrm{Tr}_B(\rho_{AB}) = \sum_{i,j,k} \alpha(i,j)\alpha^*(k,j)|i\rangle\langle k| \tag{5}$$

$$\rho_A^2 = \sum_{i',j',k'} \sum_{i,j,k} \alpha(i,j)\alpha(k,j)\alpha^*(i',j')\alpha^*(k',j')|i\rangle\langle k|i'\rangle\langle k'| \tag{6}$$

$$= \sum_{j',k'} \sum_{i,j,k} \alpha(i,j)\alpha^*(k,j)\alpha(k,j')\alpha^*(k',j')|i\rangle\langle k'| \tag{7}$$

$$\mathrm{Tr}_A(\rho_A^2) = \sum_{i,j,k,j'} \alpha(i,j)\alpha^*(k,j)\alpha(k,j')\alpha^*(i,j') \tag{8}$$

$$\tag{9}$$

For a unentangled ψ we know that $\mathrm{Tr}_A(\rho_A^2) = 1$ For ψ entangled this will be smaller than 1 (proof comes from the property of the density operator that its eigenvalues are all smaller than 1). This fits the definiton of the purity of a quantum state hence we can write:

$$P = \frac{1}{K} = \sum_{i,j,k,l} \alpha(i,j)\alpha^*(k,j)\alpha(k,l)\alpha^*(i,l) \tag{10}$$

A.2 Calculation from experimental data

A.2.1 Trace method

In the lab we can measure $|\phi(\omega_1, \omega_2)|^2$, here I outline how to extract the schimdt number from this set of values. Taking the positive square root of the matrix of values obtained from the lab you have a matrix $\mathbf{f}$ given by:

$$\mathbf{f} = \sum_{\omega_1,\omega_2} \phi(\omega_1,\omega_2)|\omega_1\rangle\langle\omega_2| \tag{11}$$

(This seems to be some weird way of writing the wavefunction as a matrix, bare with me it turns out to be useful)

$$\mathbf{f}^\dagger\mathbf{f} = \sum_{\omega_1,\omega_2,\omega_3} \phi(\omega_1,\omega_2)\phi(\omega_3,\omega_2)|\omega_1\rangle\langle\omega_3| \tag{12}$$

$$(\mathbf{f}^\dagger\mathbf{f})^2 = \sum_{\omega_1,\omega_2,\omega_3,\omega_4,\omega_5,\omega_6} \phi(\omega_1,\omega_2)\phi(\omega_3,\omega_2)\phi(\omega_4,\omega_5)\phi(\omega_6,\omega_5)|\omega_1\rangle\langle\omega_3|\omega_4\rangle\langle\omega_6| \tag{13}$$

$$(\mathbf{f}^\dagger\mathbf{f})^2 = \sum_{\omega_1,\omega_2,\omega_3,\omega_4,\omega_5,\omega_6} \phi(\omega_1,\omega_2)\phi(\omega_3,\omega_2)\phi(\omega_3,\omega_5)\phi(\omega_6,\omega_5)|\omega_1\rangle\langle\omega_6| \tag{14}$$

$$\mathrm{Tr}\left[(\mathbf{f}^\dagger\mathbf{f})^2\right] = \sum_{\omega_1,\omega_2,\omega_3,\omega_4} \phi(\omega_1,\omega_2)\phi(\omega_3,\omega_2)\phi(\omega_3,\omega_4)\phi(\omega_1,\omega_4) \tag{15}$$

I've done it this way because I wanted to figure out where the equation in [1] comes from. You can now see that equation 10 is of exactly the same form as $\mathrm{Tr}\left[(\mathbf{f}^\dagger\mathbf{f})^2\right]$ (barring the conjugates but this is okay since ϕ is real.) Taking the parallel further it can be seen that equation 12 is of the form of a reduced density matrix. Here we must make sure to normalise to make sure this is a valid reduced density matrix. The normalisation is:

$$N = \mathrm{Tr}\left[\mathbf{f}^\dagger\mathbf{f}\right] = \sum_{\omega_1,\omega_2} \phi(\omega_1,\omega_2)^2 \tag{16}$$

Giving:

$$\rho_A = \frac{\mathbf{f}^\dagger\mathbf{f}}{N} \tag{17}$$

We can then write:

$$\frac{1}{K} = \frac{\mathrm{Tr}\left[(\mathbf{f}^\dagger\mathbf{f})^2\right]}{\mathrm{Tr}\left[\mathbf{f}^\dagger\mathbf{f}\right]^2} \tag{18}$$

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "TunicsInterface.h"
#include "OsaInterface.h"
#include "PiezoInterface.h"
#include "PowermeterInterface.h"
#include "Recoupling.h"
#include "VOAInterface.h"
#include "XTA50Interface.h"

// Turn off the laser and get a blank spectrum
void getBlankSpectrum(TunicsHandle laser, OsaHandle * osa, char * name, int
    amount);

// Prints out a 2D matrix representing the joint spectrum to file
```

```
void printJointSpectrumWavelengthData(char * filename, osaRawData ** spectrum,
    char delimter, int numberOfSeedReadings);

// Take spectral scan, main loop copied from the spectral scan program
int takeSpectralScan(float lmin, float lmax, float lstep, float attenuation,
    PowermeterHandle outmeter,PowermeterHandle outmeter2, TunicsHandle laser);

// Prints out a regular spectrum to file
void printSpectrum(char * filename, osaRawData * spectrum, char delimter);

void takeAndPrintJointSpectrum(OsaHandle * osa,
                               TunicsHandle laser,
                               PowermeterHandle tapPowerMeter,
                               PowermeterHandle chipPowerMeter,
                               float osaStartWavelength_nm,
                               float osaEndWavelength_nm,
                               float osaSampleNumber,
                               float startWavelength,
                               float endWavelength,
                               float numberOfMeasurements,
                               float seedResolution,
                               int manageCoupling,
                               PiezoHandle leftPiezo,
                               PiezoHandle rightPiezo,
                               float maximumZVoltage,
                               float currentAttenuationLin,
                               int JSAsToTake,
                               int jsaNumber,
                               int tunableFilterOnCW);

XTA50System XTA50;

/* Modes of opertation
        1.  Normal JSA
        2.      Tunable Filter on CW, Normal
        3.  Normal Attenuation Scan
        4.      Tunable Filter on CW, Attenuation Scan
        5.  Tunable Filter on Pump, FWHM Scan
*/

int main (int argc, char** argv)
{

        if (argc != 31)
        {
                fprintf (stderr, "Error: Incorrect number of inputs. Expected
                    22, received %i.\n", argc-1);
                return 1;
        }

    int manageCoupling;
    float startWavelength, endWavelength, seedResolution, maximumZVoltage;
    char laserCom[256];
    char osaConnectionDescripitor[256];
    char leftPiezoCom[256];
    char rightPiezoCom[256];
```

```
    char chipPowermeterCom[256];
    char tapPowermeterCom[256];
    float cwLaserPower_mW = 0;
    float pumpLaserCurrent_mA = 0;
    float EDFACurrent_mA = 0;
    int scanType = 0;
    float startAttenuation, endAttenuation, attenuationStep;
    int cwAWGChannel, pumpAWGChannel;
    char chipName[256];
    float pumpWavelength;
    float TunableFilterCWOffset;
    float FWHMStart;
    float FWHMStep;
    float FWHMEnd;
    char tunableFilterCom[256];
    char VOACom[256];
    float ChipPowermeterAttenuation;
    float TapPowermeterAttenuation;

    sscanf(argv[1], "%f", &startWavelength);
    sscanf(argv[2], "%f", &endWavelength);
    sscanf(argv[3], "%f", &seedResolution);
    sscanf(argv[4], "%i", &manageCoupling);
    sscanf(argv[5], "%f", &maximumZVoltage);
    strcpy(laserCom,argv[6]);
    strcpy(osaConnectionDescripitor,argv[7]);
    strcpy(leftPiezoCom,argv[8]);
    strcpy(rightPiezoCom,argv[9]);
    strcpy(chipPowermeterCom,argv[10]);
    strcpy(tapPowermeterCom,argv[11]);
    sscanf(argv[12], "%f", &cwLaserPower_mW);
    sscanf(argv[13], "%f", &pumpLaserCurrent_mA);
    sscanf(argv[14], "%f", &EDFACurrent_mA);
    sscanf(argv[15], "%d", &scanType);
    sscanf(argv[16], "%f", &startAttenuation);
    sscanf(argv[17], "%f", &endAttenuation);
    sscanf(argv[18], "%f", &attenuationStep);
    sscanf(argv[19], "%d", &cwAWGChannel);
    sscanf(argv[20], "%d", &pumpAWGChannel);
    strcpy(chipName,argv[21]);
    sscanf(argv[22], "%f", &pumpWavelength);
    sscanf(argv[23], "%f", &TunableFilterCWOffset);
    sscanf(argv[24], "%f", &FWHMStart);
    sscanf(argv[25], "%f", &FWHMStep);
    sscanf(argv[26], "%f", &FWHMEnd);
    strcpy(tunableFilterCom,argv[27]);
    strcpy(VOACom,argv[28]);
    sscanf(argv[29], "%f", &ChipPowermeterAttenuation);
    sscanf(argv[30], "%f", &TapPowermeterAttenuation);

    printf("min = %f\nmax= %f\nstep = %f\n", startWavelength,endWavelength,
        seedResolution);
    VOAHandle voaHandle = InitVOA(VOACom);

    // Set up devices
    OsaHandle * osa = InitOsa(osaConnectionDescripitor);
```

```
//osaRawData * spec = osaGetSpectrum(osa);
//printSpectrum("OSASpectrum950nm.txt", spec, ' ');
//return 0;

/* CW Laser */
TunicsHandle laser = InitTunics(laserCom);
PowermeterHandle chipPowerMeter = InitPowermeter(chipPowermeterCom);
PowermeterHandle tapPowerMeter = InitPowermeter(tapPowermeterCom);

/* Piezo for side coupling */
PiezoHandle leftPiezo;
PiezoHandle rightPiezo;
if(manageCoupling == 1)
{
    leftPiezo = InitPiezo(leftPiezoCom);
    rightPiezo = InitPiezo(rightPiezoCom);
}


/* Start the tunable filter */
XTA50_SerialCom_type XTA50Vars;
int useTunableFilterForCW = 0;
if(scanType == 2 || scanType == 3 || scanType == 5)
{
    sprintf(XTA50Vars.COMPort,tunableFilterCom);
    InitXTA50(XTA50Vars);
    if(scanType == 2 || scanType == 4)
    {
        useTunableFilterForCW = 1;
    }
}

// CW LasertapPowerMeter
int numberOfMeasurements = (int)ceil((endWavelength-startWavelength)/
    seedResolution)+1;

// Get important data form OSA
float * tempArray;
tempArray = getNumericalSettings(osa,"DCA?",3); // Get wavelength range and
     sample number
float osaStartWavelength_nm = tempArray[0];
float osaEndWavelength_nm = tempArray[1];
float osaSampleNumber = (int)tempArray[2];

tempArray = getNumericalSettings(osa,"RES?",1); // Get OSA resolution
float osaResolution_nm = tempArray[0];

char *  VBW = getTextSetting(osa,"VBW?"); // Get OSA video bandwidth
char *  HDR = getTextSetting(osa,"DRG?"); // Get dynamic range mode

// Create a big file with as much information about the joint spectrum as
    possible
// Open files
FILE * infoFile = fopen("info.txt", "w");
fprintf(infoFile, "#General Information\n");
```

```
    fprintf(infoFile, "Chip : %s\n",chipName);
    // Get the time
    time_t current_time;
    char* c_time_string;
    current_time=time(NULL);
    c_time_string = ctime(&current_time);
    fprintf(infoFile, "Timestamp: ");
    fprintf(infoFile, c_time_string);
    fprintf(infoFile, "\n");
    fprintf(infoFile, "Notes : \n");
    fprintf(infoFile, "Recoupling enabled: %d\n\n", manageCoupling);
    fprintf(infoFile, "#CW Laser Parameters\n");
    fprintf(infoFile, "AWG Channel : %d\n",cwAWGChannel);
    fprintf(infoFile, "Start wavelength (nm): %f\n",startWavelength);
    fprintf(infoFile, "End wavelength (nm) : %f\n",endWavelength);
    fprintf(infoFile, "Wavelength step (nm) :  %f\n",seedResolution);
    fprintf(infoFile, "Sample number : %d\n",numberOfMeasurements);
    fprintf(infoFile, "Laser power (mW): %f\n\n",cwLaserPower_mW);

    fprintf(infoFile, "#Pump laser parameters\n");
    fprintf(infoFile, "AWG Channel : %d\n",pumpAWGChannel);
    fprintf(infoFile, "Wavelength (nm) : %f\n",pumpWavelength);
    fprintf(infoFile, "Current (mA): %f\n\n",pumpLaserCurrent_mA);

    fprintf(infoFile, "#EDFA parameters\n");
    fprintf(infoFile, "Current (mA): %d\n\n",EDFACurrent_mA);

    fprintf(infoFile, "# OSA Settings \n");
    fprintf(infoFile, "Start Wavelength (nm) : %f\n",osaStartWavelength_nm);
    fprintf(infoFile, "End Wavelength (nm) : %f\n",osaEndWavelength_nm);
    fprintf(infoFile, "Sample Number : %f\n",osaSampleNumber);
    fprintf(infoFile, "Resolution (nm) : %f\n",osaResolution_nm);
    fprintf(infoFile, "Video Bandwidth : %s",VBW);
    fprintf(infoFile, "Dynamic Range Mode: : %s\n",HDR);

    SetTunicsEmission (laser,1);
    if(useTunableFilterForCW)
    {
        SetLambda(startWavelength-0.08);
    }
    SetTunicsWavelength(laser,startWavelength);
    float initialChipPower = 10.0*log10(MeasurePowermeter(chipPowerMeter)) + 30
        + ChipPowermeterAttenuation;
    float initialTapPower  = 10.0*log10(MeasurePowermeter(tapPowerMeter))  + 30
        + TapPowermeterAttenuation;

    fprintf(infoFile,"#Powermeter initial readings\n");
    fprintf(infoFile, "Before Chip (dBm): %f\n",initialTapPower);
    fprintf(infoFile, "After Chip (dBm): %f\n",initialChipPower);
    fprintf(infoFile, "Loss (dBm): %f\n\n",initialTapPower-initialChipPower);

    system("MKDIR osaSpectrumsWithNoCW");
    system("MKDIR jointSpectrums");
    system("MKDIR powerLogs");
    //system("MKDIR spectralScans");
```

```
    /* Buffer for filenames */
    char str[500];
    // Normal JSA
    // Tunable Filter on CW, Normal
    if(scanType == 1 || scanType == 2)
    {
        getBlankSpectrum(laser,osa,"osaSpectrumsWithNoCW/noSeedSpectrumBefore"
            ,5);
        SetTunicsEmission(laser,1);
        //takeSpectralScan(startWavelength,endWavelength,seedResolution,0,
            chipPowerMeter,tapPowerMeter,laser);
        takeAndPrintJointSpectrum(osa,
                                  laser,
                                  tapPowerMeter,
                                  chipPowerMeter,
                                  osaStartWavelength_nm,
                                  osaEndWavelength_nm,
                                  osaSampleNumber,
                                  startWavelength,
                                  endWavelength,
                                  numberOfMeasurements,
                                  seedResolution,
                                  manageCoupling,
                                  leftPiezo,
                                  rightPiezo,
                                  maximumZVoltage,
                                  0,
                                  1,
                                  0,
                                  useTunableFilterForCW);
        getBlankSpectrum(laser,osa,"osaSpectrumsWithNoCW/noSeedSpectrumAfter"
            ,5);
        SetTunicsEmission(laser,1);
    }
    // Attenuation Scan
    // Tunable Filter on CW, Attenuation Scan
    if(scanType == 3 || scanType == 4)
    {
        fprintf(infoFile,"#Attenuator Settings\n");
        fprintf(infoFile, "Attenuation enabled?: %d\n",scanType);
        fprintf(infoFile, "Start Attenuation: %f\n",startAttenuation);
        fprintf(infoFile, "End Attenuation: %f\n",endAttenuation);
        fprintf(infoFile, "Attenuation Step: %f\n\n",attenuationStep);
        fprintf(infoFile,"#Attenuations\nno. watts DB Tap Power Chip Power\n");
        int numberOfJSAsToTake = (int)((endAttenuation - startAttenuation)/
            attenuationStep) + 1;
        float currentTapPower;
        float currentChipPower;
        SetDVA_attenuation(voaHandle,10*log10(startAttenuation));
        int i;
        for(i = 0; i < numberOfJSAsToTake; ++i)
        {
            sprintf(str,"osaSpectrumsWithNoCW/noSeedSpectrumBefore_%i_",i);

            getBlankSpectrum(laser,osa,str,5);
            SetTunicsEmission(laser,1);
```

```
float currentAttenuationLin = startAttenuation + attenuationStep*(
    float)i;
float currentAttenuation_dB = -10 * log10 (1-currentAttenuationLin)
    ;
SetAttenuationLin(voaHandle,currentAttenuationLin);

printf("Trying to set attenuator to %f dB\n", currentAttenuation_dB
    );
currentTapPower = 10.0*log10(MeasurePowermeter(tapPowerMeter))  +
    30 + TapPowermeterAttenuation;
currentChipPower = 10.0*log10(MeasurePowermeter(chipPowerMeter))  +
     30 + ChipPowermeterAttenuation;
fprintf(infoFile, "%d %f %f %f %f\n",i,currentAttenuationLin,
    currentAttenuation_dB,currentTapPower,currentChipPower);
//takeSpectralScan(startWavelength,endWavelength,seedResolution,
    currentAttenuation_dB,chipPowerMeter,tapPowerMeter,laser);
takeAndPrintJointSpectrum(osa,
                          laser,
                          tapPowerMeter,
                          chipPowerMeter,
                          osaStartWavelength_nm,
                          osaEndWavelength_nm,
                          osaSampleNumber,
                          startWavelength,
                          endWavelength,
                          numberOfMeasurements,
                          seedResolution,
                          manageCoupling,
                          leftPiezo,
                          rightPiezo,
                          maximumZVoltage,
                          currentAttenuationLin,
                          numberOfJSAsToTake,
                          i,
                          useTunableFilterForCW);

if(manageCoupling == 1)
{
    if(GetPiezoVoltage(leftPiezo,'z') < maximumZVoltage)
    {
        printf("Recoupling left piezo.\n");
        RecoupleDynamic(chipPowerMeter,leftPiezo,100);
    }
    else
    {
        printf("LEFT Z MAX REACHED!");
    }
    if(GetPiezoVoltage(rightPiezo,'z') < maximumZVoltage)
    {
        printf("Recoupling right piezo.\n");
        RecoupleDynamic(chipPowerMeter,rightPiezo,100);
    }
    else
    {
        printf("RIGHT Z MAX REACHED!");
```

```
                }
            }
        }
        sprintf(str,"osaSpectrumsWithNoCW/noSeedSpectrumBefore_%i_",i);
        getBlankSpectrum(laser,osa,str,5);
        SetTunicsEmission(laser,1);
    }

    //Tunable Filter on Pump, FWHM Scan
    if(scanType == 5)
    {
        for(int i = 0; i < (FWHMEnd-FWHMStart)/FWHMStep + 1; ++i)
        {
            float FWHM = FWHMStart + (float)i*FWHMStep;
            SetFWHM(FWHM);
            printf("FWHM =  %f\n", FWHM);

            sprintf(str,"osaSpectrumsWithNoCW/noSeedSpectrumBefore_%i_",i);
            getBlankSpectrum(laser,osa,str,5);
            SetTunicsEmission(laser,1);

            fprintf(infoFile, "%d %f %f\n",i,FWHM,MeasurePowermeter(
                tapPowerMeter),MeasurePowermeter(chipPowerMeter));

            takeAndPrintJointSpectrum(osa,
                                      laser,
                                      tapPowerMeter,
                                      chipPowerMeter,
                                      osaStartWavelength_nm,
                                      osaEndWavelength_nm,
                                      osaSampleNumber,
                                      startWavelength,
                                      endWavelength,
                                      numberOfMeasurements,
                                      seedResolution,
                                      manageCoupling,
                                      leftPiezo,
                                      rightPiezo,
                                      maximumZVoltage,
                                      0,
                                      1,
                                      i,
                                      useTunableFilterForCW);
            sprintf(str,"osaSpectrumsWithNoCW/noSeedSpectrumAfter_%i_",i);
            getBlankSpectrum(laser,osa,str,5);
            SetTunicsEmission(laser,1);
        }
        if(manageCoupling == 1)
        {
            if(GetPiezoVoltage(leftPiezo,'z') < maximumZVoltage)
            {
                printf("Recoupling left piezo.\n");
                RecoupleDynamic(chipPowerMeter,leftPiezo,100);
            }
            else
            {
```

```
                printf("LEFT Z MAX REACHED!");
            }
            if(GetPiezoVoltage(rightPiezo,'z') < maximumZVoltage)
            {
                printf("Recoupling right piezo.\n");
                RecoupleDynamic(chipPowerMeter,rightPiezo,100);
            }
            else
            {
                printf("RIGHT Z MAX REACHED!");
            }
        }
        fclose(infoFile);
    }

    // Record chip transmission
    fprintf(infoFile,"#Powermeter final readings\n");
    if(useTunableFilterForCW)
    {
        SetLambda(startWavelength-0.08);
    }
    SetTunicsWavelength(laser,startWavelength);
    float chip = 10.0*log10(MeasurePowermeter(chipPowerMeter)) + 30 +
        ChipPowermeterAttenuation;
    float tap = 10.0*log10(MeasurePowermeter(tapPowerMeter)) + 30 +
        TapPowermeterAttenuation;
    fprintf(infoFile, "Before Chip (dBm): %f\n",tap);
    fprintf(infoFile, "After Chip (dBm): %f\n",chip);
    fprintf(infoFile, "Loss (dBm): %f\n\n",tap-chip);
    fclose(infoFile);


    //Disable the laser
    SetTunicsEmission(laser,0);

    if(manageCoupling == 1)
    {
        //Retract the piezos (Hopefully zero is backwards)
        RampToVoltage(leftPiezo,0.00,'z');
        RampToVoltage(rightPiezo,0.00,'z');
    }

    /* Close all the things */
    ClosePowermeter(tapPowerMeter);
    ClosePowermeter(chipPowerMeter);
    if(manageCoupling == 1)
    {
        ClosePiezo(leftPiezo,0);
        ClosePiezo(rightPiezo,0);
    }
    CloseTunics(laser);
    osaWrite(osa,"SRT"); // Tell the osa to keep taking spectrums
    //CloseOsa(osa);
    CloseXTA50();
    return 0;
}
```

```
void getBlankSpectrum(TunicsHandle laser, OsaHandle * osa, char * name, int
    amount)
{
    char str[500];
    SetTunicsEmission(laser,0);
    /* Get a spectrum after experiment, might come in handy */
    printf("Taking initial spectrum without seed laser.\n");
    osaRawData * spectrumWithoutSeedLaser;
    for(int i = 0; i < amount; ++i)
    {
        spectrumWithoutSeedLaser = osaGetSpectrum(osa);
        sprintf(str,"%s%d%s",name,i,".txt");
        printSpectrum(str,spectrumWithoutSeedLaser,',');
    }
}

void printSpectrum(char * filename, osaRawData * spectrum, char delimter)
{

    FILE * fp = fopen(filename, "w");
    float * singleSpectrum = parseRawDataIntoFloatArray(spectrum,spectrum->
        osaSampleNumber,'\n');
    for(int i = 0; i < spectrum->osaSampleNumber; ++i)
    {
        fprintf(fp,"%e %e\n", spectrum->osaStartWavelength + i*(spectrum->
            osaEndWavelength - spectrum->osaStartWavelength)/(float)spectrum->
            osaSampleNumber, singleSpectrum[i]);
    }

    fclose(fp);
}

void printJointSpectrumWavelengthData(char * filename, osaRawData ** spectrum,
    char delimter, int numberOfSeedReadings)
{
    /* Print out the wavelengths to files (Probably pointless) */
    #define BUFLEN 128

    char buffer[BUFLEN];
    strcpy(buffer,filename);
    strcpy(buffer,strtok(buffer,"."));

    char osaFilename[BUFLEN];
    strcpy(osaFilename,buffer);

    char seedFilename[BUFLEN];
    strcpy(seedFilename,buffer);

    strcat(osaFilename,"_osaWavelengths.txt");
    strcat(seedFilename,"_seedWavelengths.txt");

    FILE * osaWavelengths = fopen(osaFilename, "w");
    FILE * seedWavelengths = fopen(seedFilename, "w");

    for(int i = 0; i < spectrum[0]->osaSampleNumber; ++i)
```

```
    {
        float value = spectrum[0]->osaStartWavelength + (float)i*(spectrum[0]->
            osaEndWavelength - spectrum[0]->osaStartWavelength)/(float)(spectrum
            [0]->osaSampleNumber-1);
        fprintf(osaWavelengths, "%e\n", value);
    }

    for(int i = 0; i < spectrum[0]->numberOfSeedReadings; ++i)
    {
        float value = spectrum[i]->seedWavelength;
        fprintf(seedWavelengths, "%e\n", value);
    }

    fclose(osaWavelengths);
    fclose(seedWavelengths);
}

void takeAndPrintJointSpectrum(OsaHandle * osa,
                               TunicsHandle laser,
                               PowermeterHandle tapPowerMeter,
                               PowermeterHandle chipPowerMeter,
                               float osaStartWavelength_nm,
                               float osaEndWavelength_nm,
                               float osaSampleNumber,
                               float startWavelength,
                               float endWavelength,
                               float numberOfMeasurements,
                               float seedResolution,
                               int manageCoupling,
                               PiezoHandle leftPiezo,
                               PiezoHandle rightPiezo,
                               float maximumZVoltage,
                               float currentAttenuationLin,
                               int JSAsToTake,
                               int i,
                               int tunableFilterOnCW)
{

    float currentAttenuation_dB = -10 * log10 (1-currentAttenuationLin);


    // 2 Dimensional data
    osaRawData ** spectrum = NULL;

    // Alloc the data
    spectrum = malloc(sizeof(osaRawData)*numberOfMeasurements);
    char str[256];
    sprintf(str,"jointSpectrums/jsa_%d.txt",i);
    FILE * dataFile = fopen(str, "w");
    sprintf(str,"jointSpectrums/normalised_jsa_%d.txt",i);
    FILE * normalisedDataFile = fopen(str, "w");
    sprintf(str,"powerLogs/powerLog_%d.txt",i);
    FILE * powerFile = fopen(str, "w");

    fprintf(dataFile,"xrange %e %e\nyrange %e %e\ninvertrows\n",
        osaStartWavelength_nm, osaEndWavelength_nm, startWavelength,
```

```
    endWavelength);
fprintf(normalisedDataFile,"xrange %e %e\nyrange %e %e\ninvertrows\n",
    osaStartWavelength_nm, osaEndWavelength_nm,startWavelength,endWavelength
    );

float wavelength = startWavelength;

float minsLeft = 0.0f;
for(int i = 0; i < numberOfMeasurements; ++i)
{
    int startTime = time(0);
    /* Set CW laser wavelength */
    if(tunableFilterOnCW)
    {
        SetLambda(wavelength-0.08);
    }
    SetTunicsWavelength(laser,wavelength);

    // Record the readings on the power meters
    float tapPower = MeasurePowermeter(tapPowerMeter)*100; //++20 accounts
        for the tapping
    float chipPower = MeasurePowermeter(chipPowerMeter)*(100.0/99.0);
    fprintf(powerFile,"%e,%e,%e\n",wavelength,tapPower,chipPower);
    printf("Current wavelength %.3f\n", wavelength);
    printf("Percentage: %.1f%\n", (double)100*i/(numberOfMeasurements-1));
    printf("Attenuation %.1f (dB)\n", currentAttenuation_dB);
    printf("Taking Spectrum...\n");

    spectrum[i] = osaGetSpectrum(osa);

    float normalisation = tapPower;
    if(normalisation < 0)
    {
        printf("Normalisation was negative, something is probably wrong
            with the set up!\n");
    }

    for(int j = 0; j < osaSampleNumber; ++j)
    {
        fprintf(dataFile,"%e", (spectrum[i]->floatData)[j]);
        fprintf(normalisedDataFile,"%e", (spectrum[i]->floatData)[j]/
            normalisation);
        if(j != osaSampleNumber - 1)
        {
            fprintf(dataFile, " ");
            fprintf(normalisedDataFile, " ");
        }
    }
    fprintf(dataFile,"\n");
    fprintf(normalisedDataFile,"\n");

    /* Transfer CW laser settings to the specturm struct */
    spectrum[i]->seedWavelength = wavelength;
    spectrum[i]->seedStartWavelength = startWavelength;
    spectrum[i]->seedEndWavelength = endWavelength;
    spectrum[i]->numberOfSeedReadings = (int)numberOfMeasurements;
```

```
        /* Update the wavelength for the next loop */
        wavelength += seedResolution;

        int endTime = time(0);
        minsLeft = (float)(endTime-startTime)*(float)(numberOfMeasurements-i
            +1+6)/(float)60;
        float totalTimeLeft = minsLeft*(float)(JSAsToTake-i);
        if(JSAsToTake > 1)
        {
            printf("This scan should finish in %.2f mins.\n The experiment
                should finish in %.2f mins.\n\n", minsLeft, totalTimeLeft);
        }
        else
        {
            printf("This scan should finish in %.2f mins.\n\n", minsLeft);;
        }
    }

    fclose(dataFile);
    fclose(normalisedDataFile);

    /* Data Formatting */
    printJointSpectrumWavelengthData("jsa.txt", spectrum,'\n',
        numberOfMeasurements);

    /* Freeing data and closing devices */
    for(int i = 0; i < numberOfMeasurements; ++i)
    {
        free(spectrum[i]->data);
        free(spectrum[i]->floatData);
        free(spectrum[i]);
    }

    free(spectrum);
}

int takeSpectralScan(float lmin, float lmax, float lstep, float attenuation,
    PowermeterHandle outmeter,PowermeterHandle outmeter2, TunicsHandle laser)
{
  char str[256];
  sprintf(str, "spectralScans/spectralScan%8.3f-%8.3f_att_%.3f.csv",lmin,lmax,
      attenuation);
  FILE * fp = fopen(str,"w");


  //Move to start wavelength and enable laser
  SetTunicsWavelength (laser, lmin);
  SetTunicsEmission (laser, 1);
  Sleep (100);


  //Scan
  printf ("Scanning from %8.3f to %8.3f in steps of %.3f.\n", lmin, lmax, lstep
      );
  float l = 0;
```

```
  double p1,p2;
  for (int i = 0; l < lmax; i++)
  {
    l = lmin + i*lstep;

    //SetTunicsWavelengthFast (laser, l, DWELL_TIME);
    SetTunicsWavelength (laser, l);

    p1 = MeasurePowermeter (outmeter); // Measure output
    p2 = MeasurePowermeter (outmeter2); // Measure output

    fprintf (fp, "%e, %e, %e\n", l, p1,p2);
  }

  fclose(fp);

  return 1;

}
```