NANO-SCALE CHARGE SEPARATION IN PdTeI AND PEROVSKITE PHOTOVOLTAIC DEVICES OBSERVED BY CRYSTALLOGRAPHY AND TEMPERATURE-DEPENDENT PHOTOCURRENT SPECTROSCOPY

by

Patrick Cottingham

A dissertation submitted to Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy

Baltimore, Maryland

May, 2015

Abstract

Charge separation in the solid state is a process that is fundamental to technologies such as heterogenous catalysts, batteries, and photovoltaics. Additionally, spontaneous charge separation occurs as part of certain phase transitions, such as charge ordering and the formation of charge density waves. A variety of well-developed techniques exist for studying bulk charge separation and phases in which charge separation is coherent over long lengthscales. However, many interesting charge separation processes are ordered on the nano-scale or are affected by nano-scale structure (for materials) or architecture (for devices). This thesis represents progress towards combining and improving existing methodologies in order to investigate charge separation on the nano-scale.

In Chapter 2, a combination of physical properties measurements, traditional diffraction measurements, and total scattering measurements are used to investigate a charge density wave in the material PdTeI. PdTeI features quasi-1D -Pd-Te-Pd-Te- chains with palladium formally in the 3+ oxidation state. Using pair distribution function analysis of x-ray and neutron total scattering data, we find that there is a local charge-density wave arising from the disproportionation of Pd^{3+} towards Pd^{2+} in pseudo-square planar, and Pd^{4+} in pseudo-octahedral, coordination. The magnitude and coherence length for this distortion is small, such that the average structure determined by Bragg diffraction techniques possesses higher symmetry than the local structure at all temperatures. Temperature-dependent resistivity measurements show a transport anomaly at T_{CDW} = 50 K, corresponding to the reduced fluctuations of the charge

separated Pd^{2+} and Pd^{4+} sites. At higher temperatures, the charge separation is dynamic, with local Pd^{2+}/Pd^{4+} pairs persisting up to room temperature.

Non-spontaneous charge separation may be driven by the absorption of a photon to generate one of more free carriers. A storied technique for studying this process is the measurement of photoconductivity. Contemporary technology allows for photoconductivity to be measured over a far greater continuous parameter space than previously possible. However, in order to make reproducible measurements that are comparable between instruments, it is crucially important to consider the effects of temperature, non-linearity, and the spectral width and intensity of lightsources used.

Chaper 3 describes instrumentation for photovoltage and photocurrent spectroscopy over a larger continuous range of wavelengths and temperatures than other instruments described in the literature. This instrument uses a monochromated light source with total power < 30 μW incident on the device under test to maintain low temperatures, avoid thermal artifacts, and probe different regions of non-linear responses when used in conjunction with a second light source. The instrument may also be used to measure a related property, the photomagnetoresistance. The importance of normalizing measured responses for variations in light power is discussed and a rigorous process for performing these normalizations is detailed. Several circuits suited to measuring different types of samples are described and analysis for converting measured values into physically relevant properties is provided. Chapter 3 also discusses the role of non-linearity in photoconductivity measurements.

Chapter 4 discusses results obtained by measuring the photocurrent of perovskite photovoltaic cells using the instrument described in chapter 3.

Photocurrent measurements on devices containing the perovskite $(CH_3NH_3)PbI_3$ show two distinct spectral responses when deposited in a mesoporous oxide matrix, compared with one response for planar perovskite alone. These two responses likely correspond to the medium-range ordered component and the nanocrystalline component that are known to form when the perovskite is deposited in a matrix. With a TiO_2 matrix, the shorter wavelength response has an inverted temperature dependence with increasing performance on cooling. The trend is well described by an Arrhenius-type model, suggesting that it results from thermally-activated recombination.

This work demonstrates how physical properties measurements can be used to characterize the energy- and time-scales of charge separation. It also demonstrates how crystallographic techniques can be used to examine changes in charge distribution at the nano-scale, both locally and in terms of average structure. Taken together, these results suggest a strategy for elucidating unknown mechanisms of charge separation in nano-scale systems.

Committee Members:

Prof. Tyrel McQueen

Prof. Harris Silverstone

Prof. Arthur Bragg

Acknowledgements

First and foremost I would like to thank my advisor, Prof. Tyrel McQueen. As I have traversed the steep learning curve of graduate school, Tyrel has been unfailingly patient, generous with his time, and invested in the intellectual development and well-being of all his students. In retrospect, I cannot possibly imagine a better choice of advisor.

I would also like thank Prof. Gerald Meyer, now of UNC-Chapel Hill, for some special advice and encouragement during this process. Without him this journey would likely have been impossible.

I am grateful for my interaction with all of the other members of the McQueen lab, who function not only as a peer-group but as a team and a community. I would especially like to thank Dr. Adam Phelan and Jackson. Our many mind-clearing frisbee-tosses and bike rides surely improved the quality of this work.

This work was only possible with the help of many outstanding collaborators, both within and outside the McQueen group. Within the group, Dr. James Neilson and John Sheckelton collected some of the scattering data underpinning this work. David Wallace kindly helped collect the SEM images of solar cells that are shown here. David Miller synthesized the samples of PdTeI used in Chapter 2. Dr. Ke Hu, currently of UNC-Chapel Hill, not only provided innumerable samples but also answered countless questions. Mark Koontz patiently helped collect the EDX data mentioned in this work. Dr. Mikhail Feygenson of the SNS at Oak Ridge facilitated a very pleasant first trip to a neutron facility and helped me collect the Neutron PDF data shown.

I have also had the pleasure of mentoring some outstanding undergraduates and one high school student, affectionately referred to by Kate Arpino as my 'minions'. These include Michael Good, Amanda Lemire, and Joshua Sisk. Amanda Lemire's mother, Penny, provided invaluable engineering assistance.

Finally, I would like to thank my family. I would especially like to thank my father for challenging me to explain my research over Joe Squared pizza, my mother for encouragement, and my grandmother, whose cornbread is the ultimate 'brain food'. I would also like to thank Tulia Procaccino for saying a special prayer that I might be accepted to Johns Hopkins University.

Table of Contents

Table of Figures

Chapter 1

Chapter 2

Chapter 4

Table of Tables

Chapter 1: Introduction to Charge Separation at the Nano-scale

1.1 Introduction

Charge separation is a displacement of charge density, either between crystallographically distinct atoms, or across an interface. It may occur spontaneously or in response to some external stimulus such as the absorption of a photon or an applied bias. In materials, charge separation may occur spontaneously during a phase transition. Often, this takes the form of the ionization of impurities,[1] charge ordering (CO), or the formation of a charge density wave (CDW).

Charge separation in response to an external stimulus is fundamental to a host of technologies that underpin modern life. Solid-state transistors function by changing the width of a depletion region through oxidizing and reducing impurity ions on either side of a p-n junction. In first-generation solar cells, current is generated by creation of free carriers upon the absorption of a photon, followed by separation of the carriers across a p-n junction.

Many well-developed techniques exist for studying bulk charge separation and the ordering of charge separation over long length scales. Techniques in the former category include capacitance spectroscopy, *I-V* measurements, and measurements of photoconductivity as well as nuclear magnetic resonance (NMR), electron paramagnetic resonance (EPR), ultra-violet photon spectroscopy (UPS), and x-ray photon spectroscopy (XPS). Techniques in the latter category include traditional diffraction crystallography, using either x-rays or neutrons.

In many systems, charge separation occurs on, or is ordered over, length scales in the nano regime. Many CO and CDW phases are coherent over less than

one micron, especially in materials with substantial numbers of defects.[2,3,4] The performance of devices such as the actively researched lead-halide perovskite solar cells often depends heavily on nanostructured interfaces where charge separation occurs. Elucidating phenomena that occur on this length-scale requires using a combination of existing techniques for measuring bulk properties of materials together with probes of local structure such as pair-distribution function (PDF) analysis of total scattering data. It also warrants the further development of existing techniques such as temperature-dependent measurements of photoconductivity which, when combined with local-structure information, can shed light on charge separation processes. This thesis presents progress towards those ends.

1.2 Spontaneous Charge Separation in Materials

Charge ordering transitions involve a localization of mobile charges that gives rise to a long-range ordering of ions in unlike oxidation states. The transition to a CO state was first observed in magnetite, Fe_3O_4, the Verwey transition. The transition was observed as a discontinuity in the material's resistance upon cooling below the transition temperature $T_{CO} \approx 120$ K,[5] which is also accompanied by changes in the magnetic susceptibility and heat capacity. However, the structure of Fe_3O_4 was not determined for nearly a century after the observation of the transition, owing to the complexity of the CO structure featuring Fe-Fe-Fe 'trimerons'.[6] Several simpler CO motifs, including quasi-two dimensional 'checkerboards'[7] and stripes,[8,9] also occur. The change in resistance upon cooling through the CO transition results from the elimination of conduction by electron hopping. Ordering occurs when the

kinetic energy of hopping electrons is less than the minimization of electrostatic potential achieved by ordering. CO can therefore be described as a local, rather than collective electronic behavior. Fig. 1a is a generalized depiction of charge ordering.

A less extreme form of charge separation, which does not necessarily involve complete disproportionation of like atoms, is the formation of charge density waves (CDWs). CDWs are also fundamentally different from CO states in that CDW order is driven by collective electronic behaviour. This behaviour was theorized by Sir Rudolph Peierls in 1931 who derived that any one dimensional (1D) chain of atoms will form a periodic modulation in electron density, accompanied by the opening of a gap and displacement of the ions (Fig. 1b).[10]

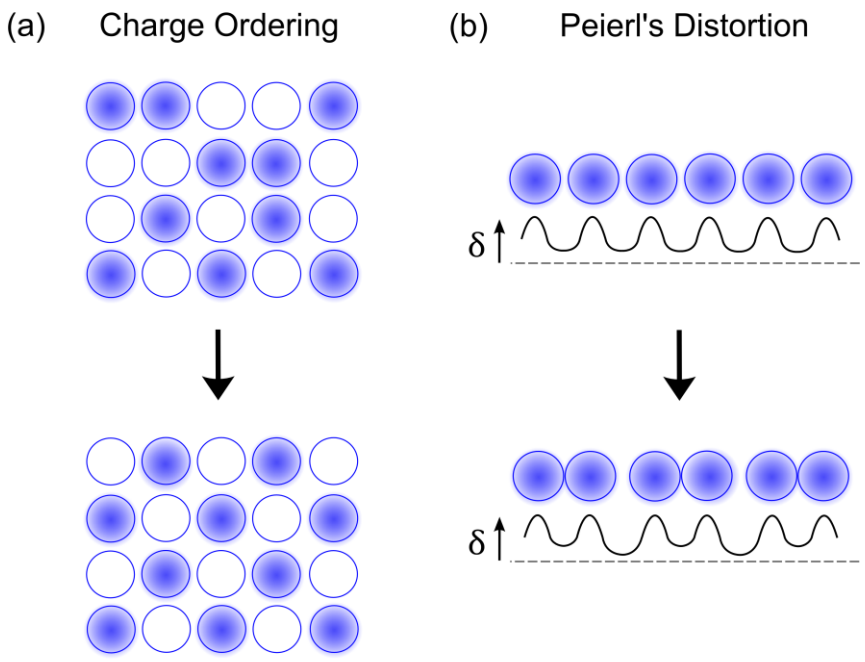


Fig. 1.1 A schematic representation of **(a)** charge ordering and **(b)** the formation of a charge density wave in a 1D chain. Blue and white spheres represent atoms in unlike oxidation states. The solid black line in **(b)** indicates charge density.

CDWs have been observed in a variety of quasi-one dimensional[11] (1D) and quasi-two dimensional[12] systems as well as in three dimensional[13] materials. 1D materials in particular are expected to form CDWs and several forms of these have been observed.[14,15,16] In the case of chains of edge-sharing transition metal polyhedra, metal-metal bonding is observed.[14] For linear -M-X-M-X- chains (abbreviated -M-X-M-) wherein M is a transition metal and X is a halogen, either a spin-Peierls distortion[15], in which magnetic exchange interactions drive a modulation of M-X bond distances, or actual charge disproportionation of the metal[16] is observed.

Fig 1.2a depicts changes in the real-space distribution of energy levels below and above a CO transition. Below T_{CO}, eletrons are ordered in a fashion that minimizes the electrostatic potential energy and the movement of individual charges between adjacent sites requires an increase in energy. Above T_{CO}, electron kinetic energy is sufficient to allow hopping, and the disordered distribution of charge gives rise to a continuum of states. In contrast, CDW formation is driven by changes in reciprocal space (Fig. 1.2b). The periodic modulation in electron density depicted in Fig. 1.1b leads to a folding of the Brillouin zone at $k_{CDW} = 2\pi/a$. It is the lowering of the Fermi energy by opening a gap at $k = k_{CDW}$ that drives CDW formation.

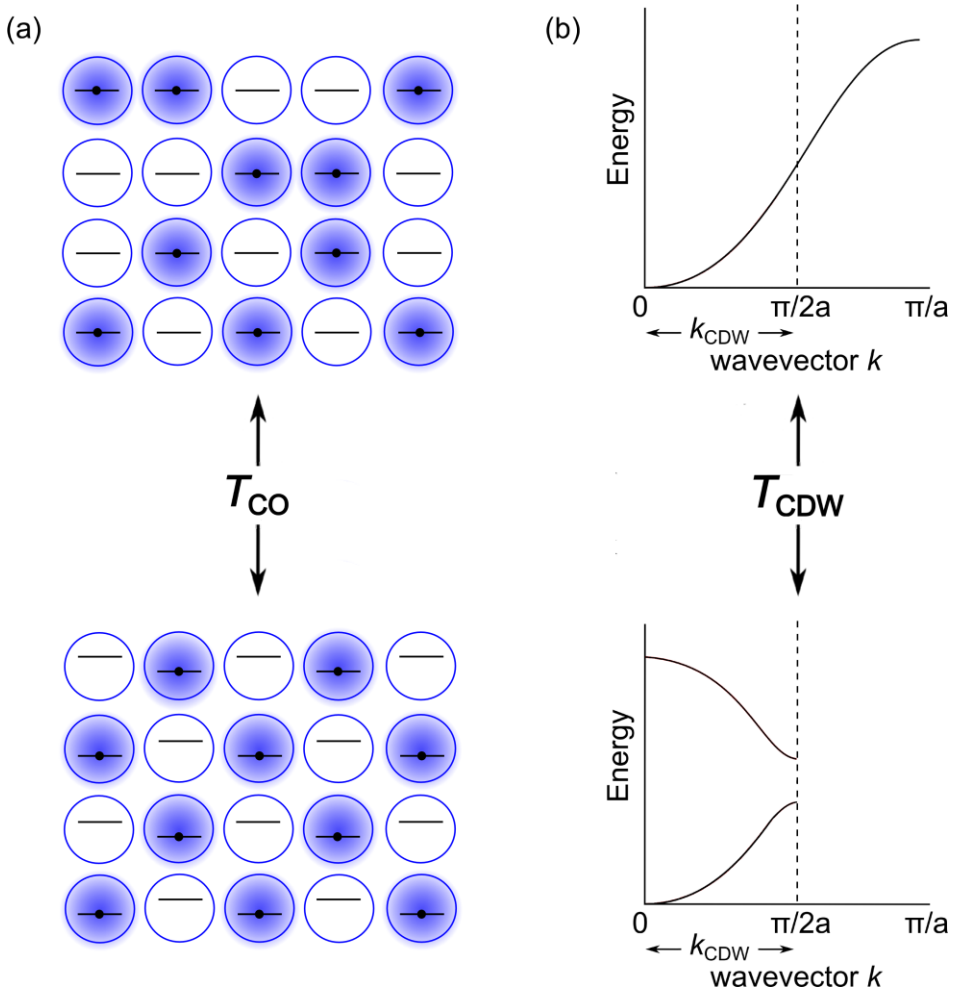


Fig. 1.2 (a) The distribution of energy levels in real space above and below a CO transition **(b)** The dispersion relation of a material above and below a transition to a CDW phase with wavevector $k_{CDW} = \pi/2a$

An additional distinction between CO and CDW phases is that k_{CDW} is not necessarily a sum of integer multiples of the reciprocal lattice vectors. If this condition is satisfied the CDW is described as 'commensurate' and otherwise is it 'incommensurate'. Incommensurate CDWs can contribute to conduction in a process sometimes referred to as 'sliding',[17] although the physical mechanism of CDW sliding remains unclear. Dynamic CO states have been observed as well.[18]

1.3 Observation of Charge Separation by Scattering Techniques

1.3.1Bragg Diffraction

When radiation from a point source impinges on a crystal and is scattered towards a point detector, rays which scatter off of different lattice planes will experience a difference in path length (Fig. 1.3). The differences in pathlength for scattered particles will lead to interference patterns at the target, with the condition for maximum constructive interference given by Bragg's law[19]:

$$2d \sin\theta = n\lambda \qquad (\mathbf{1.1})$$

where d is the spacing of the lattice planes, θ is the scattering angle, λ is the wavelength of the incident radiation, and n is a positive integer accounting for scattering from lattice planes with spacings that are multiples of λ. One point of confusion that often results from the heuristic picture of diffraction given in Fig. 1.3 is the suggestion that diffraction occurs between two incident particles of the same phase. Rather, the intensity of most incoherent radiation sources is such that particles interact with the material one at a time and the resulting diffraction patterns are the result of a particle of radiation interfering with itself.

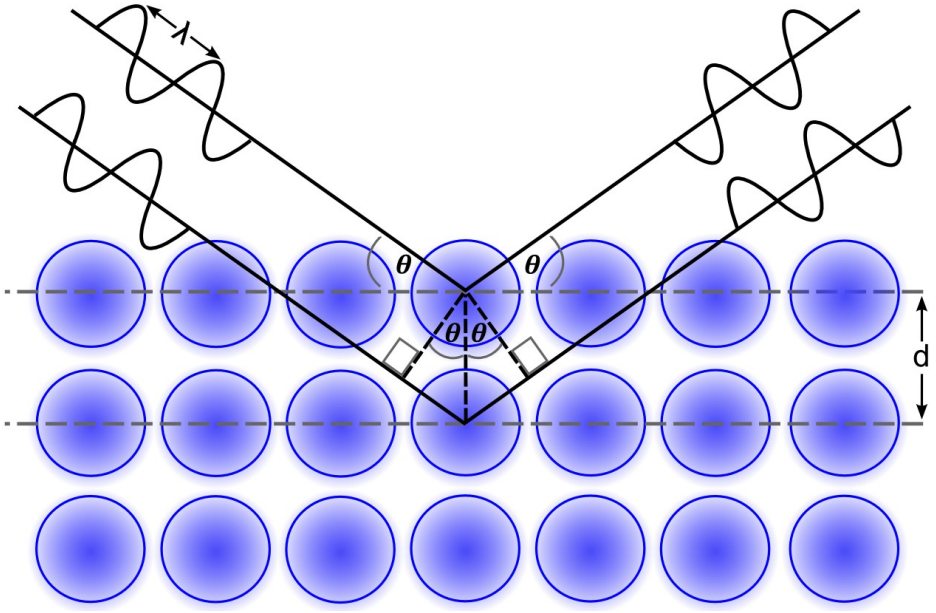


Fig. 1.3 A schematic depicting diffraction by a crystal

The two most common variations of diffraction measurements are *single crystal* and *powder* diffraction. The single crystal technique is advantageous for studying the crystal structure of materials because the diffraction patterns collected are a direct picture of the crystal structure in reciprocal space. However, for the purposes of studying charge separation powder diffraction is preferable not only because large single crystals may not be available, but also because a single crystal may contain multiple CO or CDW domains, resulting in a complicated image.

In powder diffraction experiments samples are ground into a fine powder and spread onto a flat sample stage. In the limit of very many small crystals oriented randomly relative to the radiation source, the incident radiation will sample all possible crystal orientations. In a typical laboratory powder diffractometer (Fig. 1.4), the source is fixed and the sample stage and detector are rotated and swept, respectively, in order to record over a range of scattering angles. When the scattering

angle satisfies the Bragg condition for some set of lattice planes, diffraction peaks also known as *reflections* will be detected, except in the case where out-of phase scattering by other lattice planes or atoms inside a unit cell interferes destructively leading to a *systematic absence.*

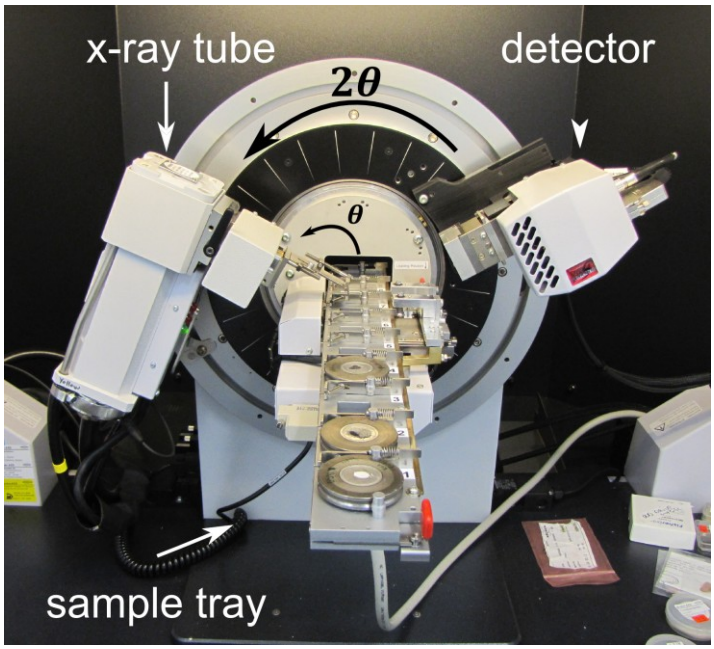


Fig. 1.4 A typical laboratory diffractometer

The unit cell of a material may be determined from the angles (typically 2θ, the angle travelled by the detector is used) at which reflections occur. Well-ordered CO and CDW phases may be detected by the appearance of *superstructure* or *supercell* reflections. These are typically weaker reflections that, when indexed with the lattice parameters of the unit cell above the transition temperature, have *h, k,* and *l* values that are 0, 1, or 1/x where x is an integer, reflecting enlargement of the unit cell due to ordering.

In order to determine the *crystal structure* of a material, it is necessary to consider other aspects of a diffraction pattern such as the shape and intensity of reflections. For x-ray diffraction of a perfect crystal, the integrated intensity of a reflection is described by

$$I_{hkl} = |F_{hkl}|^2 \qquad (\mathbf{1.2})$$

where F_{hkl} is the structure factor given by

$$F_{hkl} = \sum_{j=1}^{N} f_j \left(\frac{\sin\theta_{hkl}}{\lambda}\right) e^{2\pi i(hx_j+ky_j+jz_j)}. \qquad (\mathbf{1.3})$$

In Eqn. 1.3, j is an index for the atoms in the unit cell, x_j, y_j, and z_j are the fractional coordinates of the atom, and f_j is a *form factor* that describes the ability of a particular atom to scatter x-rays. When neutrons are used, f_j is replaced with a *neutron scattering length*, b_j, which depends not only on the particular element but also the specific isotope. f_j depends strongly on the scattering angle, but b_j is independent of θ.

Automated data collection allows for thousands of data points to be collected easily. For this reason, any model that attempts to fit a diffraction data set analytically using a reasonable number of variables would be hugely overdetermined. Instead, algorithmic methods are used to fit models to diffraction data, and those models are evaluated on the basis of statistical measures of goodness-of-fit. By far the most widely used method for fitting powder diffraction data is Rietveld refinement, which uses a least-squares approach to minimize the difference between calculated and observed diffraction patterns, while varying parameters such as atom positions and occupancies, thermal displacement parameters, and various instrumental parameters.[20]

1.3.2 PDF Analysis of Total Scattering Data

The heuristic explanation of Bragg diffraction in section *1.3.1* supposes that samples consist of infinitely large, perfect crystals. In principle, only one atom is needed for diffraction to occur, although reflections broaden as crystal size is reduced as shown in Fig. 1.5 (all commonly used structure fitting methods account for this broadening). All real samples also include features that are not ordered over long ranges, and these appear as broad contributions to the background of diffraction patterns. These features are referred to as the *local structure* of a material and include the disordered displacement of ions in ferroelectric materials above the Curie temperature and defects such as interstitial atoms. The techniques that use the broad background of scattering patterns to study local structure are referred to as *total scattering* techniques.

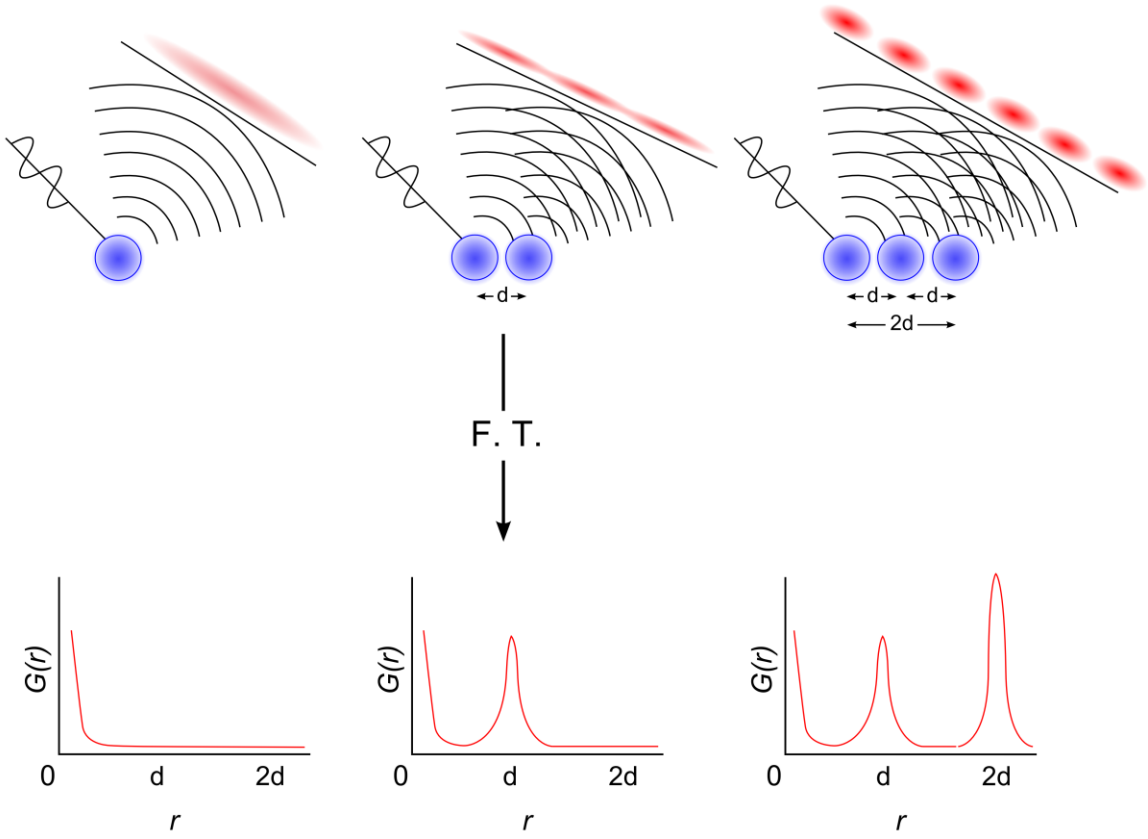


Fig. 1.5 A schematic depicting PDF analysis of total scattering data. Scattering from a small number of of atoms creates a highly broadened diffraction pattern. Applying a Fourier transform to this pattern yields the function *G(r)* in which the integrated intensity of peaks is proportional to the number of pairs of atoms with a given interatomic spacing. The intensity of scattering from a single atom is proportional to sinc(θ), but because the central maxima is much more intense that other maxima it is depicted here as a single, diffuse peak.

The most developed technique for studying local structure using total scattering data is pair distribution function (PDF) analysis. The data analyzed in PDF is the *total scattering structure function*, $S(Q)$, where Q is the scattering vector given by

$$Q = \frac{4\pi \sin\theta}{\lambda} \ . \qquad (\mathbf{1.4})$$

$S(Q)$ is obtained from the measured intensity of coherent scattering $I(Q)$, corrected for instrumental factors, polarization and the Q-dependence of form factors (in the case of x-rays) or scattering lengthss (in the case of neutrons).

When the following Fourier transform is applied to $S(Q)$, the resulting function $g(r)$ is the pair-distribution function which describes the average distance between pairs of atoms in a sample, including those related to defects:

$$g(r) = \frac{1}{\pi^2 r \rho_0} \int_0^\infty Q[S(Q) - 1] \sin(Qr)\, dQ \ . \qquad (\mathbf{1.5})$$

The function $g(r)$ may be fit with models of the local structure of a material in order to gain information not provided by diffraction.

1.3.3 Radiation Sources

Diffraction measurements can be performed with many types of radiation but x-rays and neutrons are most commonly used for structure determination. The combination of x-ray and neutron datasets is particularly useful because the form factors and scattering cross-sections for x-rays and neutrons, respectively, are substantially different for many elements. The comparison of x-ray and neutron data can help with the correct identification of the element on a given crystallographic position. The use of neutrons can be disadvantageous for identifying rapidly fluctuating distortions for reasons related to the energy-time formulation of the uncertainty principle. If the period of a distortion is comparable or smaller than the uncertainty in the time-of-arrival of a neutron, then the distortion may not be resolved.

The resolution of diffraction measurements is limited by the wavelength of the radiation source used. For rapid identification of samples, benchtop x-ray diffractometers are routinely used. The most common radiation source in these instruments is an x-ray tube in which electrons are accelerated by an electrical potential and impact a metal target. When the accelerated electrons interact with a core electron, causing it to be ejected, electrons from higher orbitals may relax to the vacated orbital emitting x-rays with a characteristic energy in the process. The most commonly used emission line is the K_α of copper, which has a wavelength of $\lambda = 0.154$ nm.

For higher resolution x-ray diffraction, synchrotron sources are needed. Synchrotrons use magnets to accelerate charged particles (typically electrons) to nearly relativistic speeds and contain them within an evacuated torus. A fraction of these particles are then released into an *insertion device*. For the purposes of generating x-rays for crystallography, insertion devices typically consist of an array of permanent magnets. When relativistic electrons enter the insertion device, they are forced to oscillate by the magnetic field, generating intense, coherent x-rays.

Two technologies are commonly used to supply neutrons for scattering experiments. *Spallation* sources generate neutrons through the collision of highly accelerated charged particles with a heavy metal target, leading to the expulsion of neutrons. Alternately, neutrons can be produced in a nuclear reactor via fission processes. Both of these types of sources generate "white" neutrons with a distribution of energies. Neutrons are fundamentally different from photons in that their velocity is not fixed at the speed of light, but rather is related to the wavelength

via the de Broglie relation,

$$\lambda = {}^{h}/_{p}, \tag{1.6}$$

where λ is the wavelength of the neutron, *h* is Plank's constant, and *p* is the momentum of the neutron.

The distribution of neutron velocities allows for *time-of-flight* (TOF) instruments to be constructed. In a TOF instrument, rather than changing the scattering angle in order satisfy the Bragg condition for different *d*-spacings, the measurement geometry is fixed and scattering intensity is measured at different times. Because the time required for a neutron to travel the distance from the source to the sample is related to its wavelength, the wavelength of neutrons reaching the sample at time *t* after the release of a pulse of neutrons is known. The change in scattering intensity with time can be used to determine which *d*-spacings satisfy the Bragg condition. Alternately, a neutron beam may be monochromated using a specialized diffraction grating and this monochromatic beam may be used in an instrument design similar to those using x-rays.

For structural determination, most crystallographic techniques rely on *elastic scattering,* scattering in which no energy is transferred between particles of radiation and atoms in the sample. Both x-rays and neutrons may also be scattered *inelastically*, and inelastically scattered radiation which is not filtered by some means of energy discrimination will appear as background in diffraction measurements. Techniques which measure inelastic neutron scattering can determine, among other things, the phonon density-of-states (PDOS) of materials. *Resonant Inelastic X-ray Scattering* (RXIS) is a technique which measures inelastically scattered x-rays in

order to gain information about periodic modulation of the electronic structure of atoms.

1.4 Measurements of Photoconductivity

1.4.1 Carrier Generation

For a given wavelength of light λ, the intensity of light arriving at a certain depth within a sample, *I(x, λ)*, is given by the Beer-Lambert law,

$$I(x,\ \lambda) = I_0(1 - R(\lambda))e^{-\alpha(\lambda)x}, \quad \textbf{(1.7)}$$

where I_0 is the light intensity arriving at the surface of the sample, $R(\lambda)$ is the reflectivity of the material at that wavelength, α is the absorption coefficient of the material, and x is the depth within the sample. Carriers may be generated by the excitation of an electron from the valence band to the conduction band, in which case both a free electron and a free hole are generated (Fig. 1.6a). Alternately, a lower-energy transition involving excitation from an in-gap defect state can generate one free carrier and another which remains trapped on the defect site. Either of these processes may generate free carriers directly or through the formation of coulombically bound excitons (electron-hole pairs) which may dissociate to form free carriers. For an ideal semiconductor or insulator under illumination with wavelength λ and with thickness t in the limit of low intensity the rate of carrier generation $f(\lambda)$ depends linearly on the incident light power according to:

$$f(\lambda) \propto \int_0^t I(x,\lambda)dx = I_0\,. \quad \textbf{(1.8)}$$

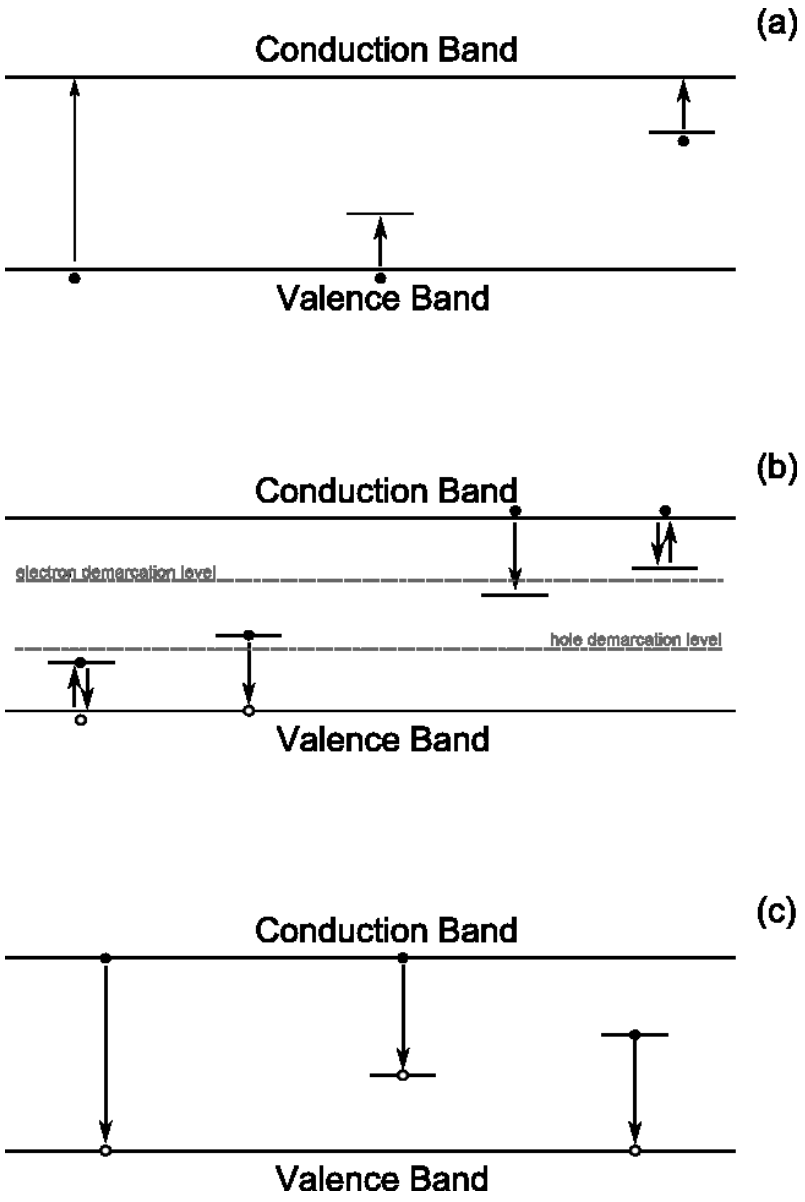


Fig. 1.6 (a) Absorption of a photon can promote electrons from the conduction band to the valence band, from the valence band into an in-gap defect state, or from an in-gap defect state into the conduction band. **(b)** The electron and hole demarcation levels indicate the energies at which an electron or hole is equally likely to be thermally excited back into the band, or to recombine **(c)** Direct recombination occurs between a free hole in the valence band and a free electron in the conduction band. Recombination can also occur involving one free carrier and one captured carrier at a recombination center.

Under constant illumination, the sample will eventually reach a steady state where the rate of optical generation of carriers is equal to the rate of recombination. That is to say, the change in carrier concentration upon illumination will be a function of the rate of free carrier generation and the free carrier lifetime. If one makes the simplifying assumption that the changes in free electron concentration, Δn, and free hole concentration, Δp, are homogenous throughout the illuminated region, then the change in carrier concentration can be expressed with

$$\Delta n = f(\lambda)\tau_n. \quad \textbf{(1.9)}$$

where τ_n is the lifetime for free electrons and

$$\Delta p = f(\lambda)\tau_p. \quad \textbf{(1.10)}$$

where τ_p is the lifetime of free holes. This assumption approaches technical correctness in the limit of very thin samples or for materials with long carrier lifetimes and long diffusion lengths, in which the diffusion of carriers will smooth gradients in carrier concentration arising from the eqn. 1.7.

Illuminating a sample will affect the conductivity of the sample according to

$$\Delta\sigma = f(\lambda)e\left(\mu_n\tau_n + \mu_p\tau_p\right). \quad \textbf{(1.11)}$$

where $\Delta\sigma$ is the change in conductivity upon illumination (the photoconductivity), e is the electron charge, and μ_n and μ_p are the free electron mobility and the free hole mobility, respectively .

In the limit of very small changes in $f(\lambda)$,μ_n, μ_p, τ_n, and τ_p are approximately constant, so it is possible to determine relative changes in $f(\lambda)$ by measuring changes in $\Delta\sigma$ while scanning λ. Abrupt changes in $f(\lambda)$ while scanning λ indicate band edges or the

energies of transitions to and from in-gap defect states. For a material in which $f(\lambda)$ is well-known, measurements of $\Delta\sigma$ can be used to calculate τ_n and τ_p.

1.4.2 Recombination

Recombination describes the processes by which free carriers are eliminated. When a system is in a steady state, the rate of recombination will be equal to the rate of free carrier generation from all sources (optical, thermal, etc.). The rate of recombination determines the lifetime of free carriers in a material. For very high rates of carrier generation or in extremely pure materials, *direct recombination* between free electrons and free holes can cause a measureable reduction of the photoconductivity.

Free carriers may also relax into in-gap defect states known as *traps* and *recombination* centers. Traps are in-gap states which are close in energy to the bands. Free carriers which relax into these states are rapidly thermally re-excited into the bands. Trapping decreases the carrier mobility, but not the lifetime. Recombination centers are in-gap states that are farther away in energy, in which captured carriers are more likely to recombine with a free carrier of opposite charge. The *electron demarcation level* and *hole demarcation level* indicate the energy at which a carrier is equally likely to remain trapped or to recombine and distinguish between traps and recombination centers. Both direct recombination and recombination of captured carriers may be either radiative, involving the emission of a photon, or non-radiative, in which case the energy difference contributes to the thermal energy of the lattice.

1.4.3 Non-linearity

The photoconductivity of many materials is non-linear. At low irradiances $\Delta\sigma$ is suppressed because the trapping of carriers by defects reduces their mobility (Fig. 1.7a).

As I_0 increases, the majority of traps will eventually become occupied due to the increasing number of free carriers. As I_0 increases further the photoconductivity may become approximately linear due to the saturation of traps. As I_0 increases even further, the rates of direct recombination and multi-particle scattering increase, decreasing τ and μ, respectively. These effects suppress $\Delta\sigma$ at higher I_0.

In order to isolate the contribution of the non-linear effects described above to the overall photoconductivity, it is useful to measure the effect on σ of a small perturbation to a larger incident intensity. This technique is illustrated in Fig. 1.7b where a continuous light source, I_{DC}, is used to place the photoresponse in the desired point on the photoconductivity response curve, with a frequency-modulated perturbation ΔI_{AC}. As ΔI_{AC} becomes very small, measurements of $\Delta\sigma$ will approximate the effect of a single additional excitation superimposed on a given $f(\lambda)$, i.e. in the instantaneous slope of the response function at I_{DC}. This can be useful for measuring specifically the "intrinsic" photoconductivity of a material in the absence of non-linear effects, or to measure the effects of trapping, multiple carrier scattering, etc. I_{DC} and ΔI_{AC} need not necessarily be of the same wavelength. In fact, it is advantageous to apply I_{DC} and ΔI_{AC} of different wavelengths if one wishes to measure, for example, rates of de-trapping at different quasi-Fermi levels.

Deviations from the assumption stated in section 1.4.1, that the change in carrier concentration is homogenous throughout the sample, are less important for samples which have a large linear region in their photoconductivty. Even for a material with short carrier lifetimes and short diffusion lenghts, gradients in Δn and Δp will not affect light-

induced changes in physical properties if the response of the sample is linear over the range from $0 \leq I(x, \lambda) \leq I(0, \lambda)$.

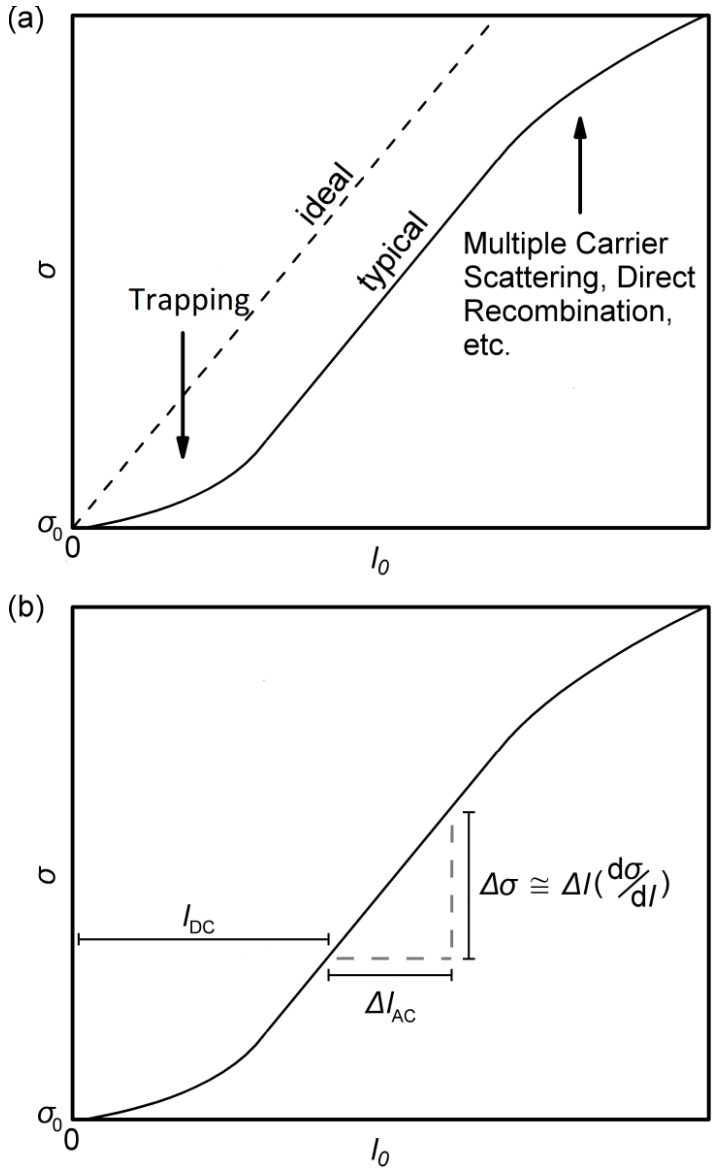


Fig. 1.7 (a) The dashed line indicates the conductivity σ as a function of incident light intensity I_0 for an ideal material. The solid line indicates the same function for a typical photoconductor. **(b)** One intense and constant light source (I_{DC}) and a smaller, modulated light source (I_{AC}) are used to select a specific region of the photoconductive response.

1.4.4 Direct and Indirect Gaps

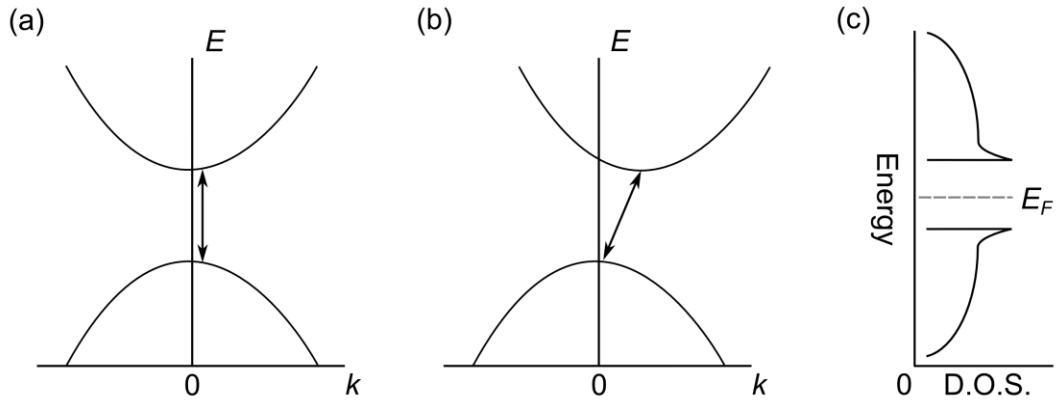


Fig. 1.8 Dispersion relations for **(a)** a direct-bandgap material and **(b)** an indirect bandgap material. The double-headed arrow represents the excitation and relaxation of carriers accross the gap. **(c)** shows a representative density-of-states for a semiconductor or insulator. The dashed line indicates the Fermi level E_F.

Semiconductors and insulators are classified as having either a *direct* or an *indirect* bandgap and this distinction has important consequences for photoconductivity, especially as a function of temperature. The dispersion relation *dE/dk*, where *E* is energy and *k* is crystal momentum, is shown in Fig. 1.8a for a direct bandgap material and in Fig. 1.8b for an indirect bandgap material. For a direct bandgap material the conduction band minimum and valence band maximum occur at the same point in *k*-space, whereas for an indirect bandgap material they occur at different *k* values. Because *k* is conserved, the lowest-energy excitation in a direct bandgap material occurs without interaction with a lattice vibration, or *phonon*. For an indirect bandgap material the lowest energy transition involves the creation or destruction of a phonon combined with a change in *k* for the excited electron. This interaction imparts a temperature-sensitivity to the photoconductivity spectra of indirect bandgap materials, as the probability of a excitation

at a given energy will be proportional to the population of phonons with the appropriate *k*. In photoconductivity spectra of indirect bandgap materials, the temperature-dependence typically manifests as a narrowing of the spectrum and a decrease in the integrated intensity upon cooling..

The probability of a given transition is also proportional to the *density of states* (DOS) at the relevant energies. A typical density of states for a semiconductor or insulator is shown in Fig. 1.8c. Because the DOS is inversely proportional to the dispersion relation, *dE/dk*, there is typically a maximum in the DOS near the band edge. This maximum in the DOS may lead to a local maximum in photoconductivity spectra near the long-wavelength photoconductivity edge.

1.4.5 Other Mechanisms of Photoconductivity

In more complex systems, photoresponses may arise due to processes other than the excitation of carriers across a gap. In polarizable materials, local dipoles may orient in response to coulombic attraction to a carrier, causing the carrier to become trapped in an in-gap *polaronic* state. The absorption of sub-bandgap photons may subsequently excite these carriers from the polaronic state to the conduction band, causing materials to display a long-wavelength photoresponse. This effect is particularly pronounced in organic photovoltaics (OPVs) and has been studied by various *pump-push* and *pump-push-probe* spectroscopies.[21,22,23]

Materials which host a charge-density wave have been shown to possess non-linear photoresponses. These responses have been attributed to a combination of free carrier generation and light-induced changes in the collective behavior of the CDW state.[24,25] A more extreme form of conductivity due to light-induced changes in

collective electronic behavior is transient light-induced superconductivity observed in $La_{1.675}Eu_{0.2}Sr_{0.125}CuO_4$.[26]

More rarely, materials may exhibit *negative photoconductivity*. In the simple case of doped semiconductors, this has been attributed to the excitation of minority carriers from defects, which then recombine with majority carriers.[27] Superconductivity may also be destroyed in some materials by exposure to intense light.[28]

1.4.6 Measurement and Normalization

Photoconductivity can be measured by applying a constant current across a sample and measuring changes in voltage upon illumination, by applying a constant voltage and measuring changes in current. Measurements of the change in the Hall voltage upon illumination, the *photo-Hall* effect, can also be used to calculate changes in carrier concentration. Examples of several circuits for the measurement of photoconductivity are discussed in section 3.3.

Given the dependence of $f(\lambda)$, and thus $\Delta\sigma$, on the incident light intensity, it is desirable to compute a rate of carrier generation normalized by incident light intensity, $f^*(\lambda)$, which is an intrinsic property of the sample under measurement according to

$$f^*(\lambda) = f(\lambda)/I_0. \quad \textbf{(1.12)}$$

The function $f^*(\lambda)$ computed in this way still contains factors related to reflection of light by the sample and transmission of light through the sample. Accurate determination of these factors involves measurements of a sample's reflectance and transmittance under the same conditions as the photoconductivity measurement. However, it is not desirable in many instances to correct for reflection and transmission given that these are intrinsic (non-instrumental) qualities of all real devices (and their applications). Typically, the

value used for normalization will be I_0, which should be computed from real-time measurements for each λ in order to correct for fluctuations in the light sources' output over time. Corrections for attenuation by elements in the optical path in order to compute the actual I_0 impinging on the sample are discussed below.

If the earlier assumption that carriers are generated homogenously throughout the illumination region holds, then I_0 can also be defined interchangeably as either an intensity (power per unit area) or a power. For instruments with an illumination area smaller than the device under test, or that illuminate a sample unevenly, it is more useful to define I_0 in units of power.

1.5 Other Physical Properties Measurements

1.5.1 Magnetic Susceptibility

Charge separation may be observed indirectly by measuring changes in the magnetic susceptibility of a sample. Magnetic susceptibility is defined as

$$\chi = \frac{\partial M}{\partial H} \qquad (\mathbf{1.13})$$

where χ is the susceptibility and $\partial M/\partial H$ is the change in the magnetization of the sample with a change in the magnetic field. Charge separation is commonly accompanied by changes in the susceptibility during transistions to CO and CDW phases due to changes in the pairing of electrons or the coordination environment of ions.

Two commonly used techniques of measuring the magnetic susceptibility are shown in Fig 1.9. In both techniques the sample is suspended by a non-magnetic tube or rod inside a set of magnetic coils. In AC measurements of magnetic

susceptibility an oscillating magnetic field is applied to the sample by applying an alternating current to the outer coil (not shown). Changes in the magnetization of the sample with the changing field are measured by recording the current that is induced in a seperate measurement coil.

In a DC magnetic susceptibility measurement, a constant external field is applied. The sample is then periodically displaced relative to the measurement coil in the direction of the magnetic field using a motor. As before, the magnetization of the sample is determined by measuring the current induced in a measurement coil.

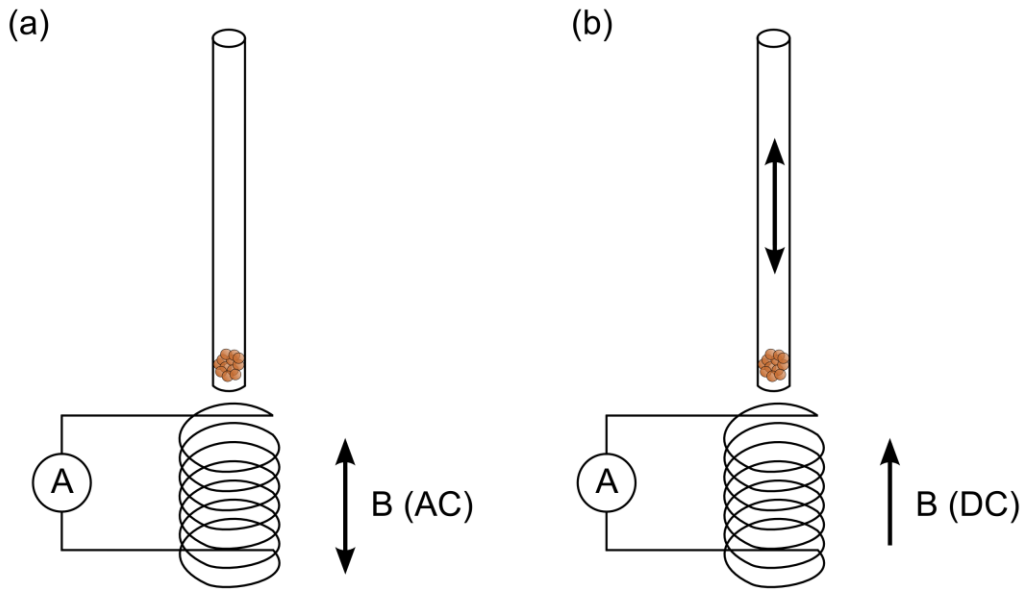


Fig 1.9 (a) AC and **(b)** DC measurements of magnetic susceptibility

1.5.2 Heat Capacity

Any charge separation that results in a loss of electronic entropy will be accompanied by a decrease in the heat capacity of the sample. The effects of charge separation on heat capacity are especially pronounced for transitions to CO and

CDW phases where the long-range ordering of charges often results in an observable decrease in the heat capacity.

A commonly used technique for measuring the heat capacity of materials and devices is the *semi-adiabatic pulse technique*. In this technique the sample is mounted on sample stage which is suspended by thin wires in order to minimize thermal contact with it's surroundings (Fig. 1.10a). The stage contains a resistive heating element and thermocouple and thermal contact between the sample stage is increased using a thermally conducting hydrocarbon grease.

During a measurement a known amount of heat, $q_{in,\,joule}$, is applied using the heating element and the temperature of the sample is monitored over time using the thermocouple. The maximum temperature reached by the sample and the rate of cooling can be fit by a function which has as parameters the heat capactity of the sample and the rate of heat loss to the surroundings. Typically, the thermal conductivity of the suspending wires is well-characterized and the heat transferred to the instrument via the wires, $q_{out,\,wires}$, is assumed to be much greater than the heat transfered conductively to the atmosphere in the sample chamber, $q_{out,\,atm}$ or the heat transfered radiatively to the surroundings, $q_{\mathrm{out,\,rad}}$. The heat balance for a measurement is shown in Fig. 1.10b.

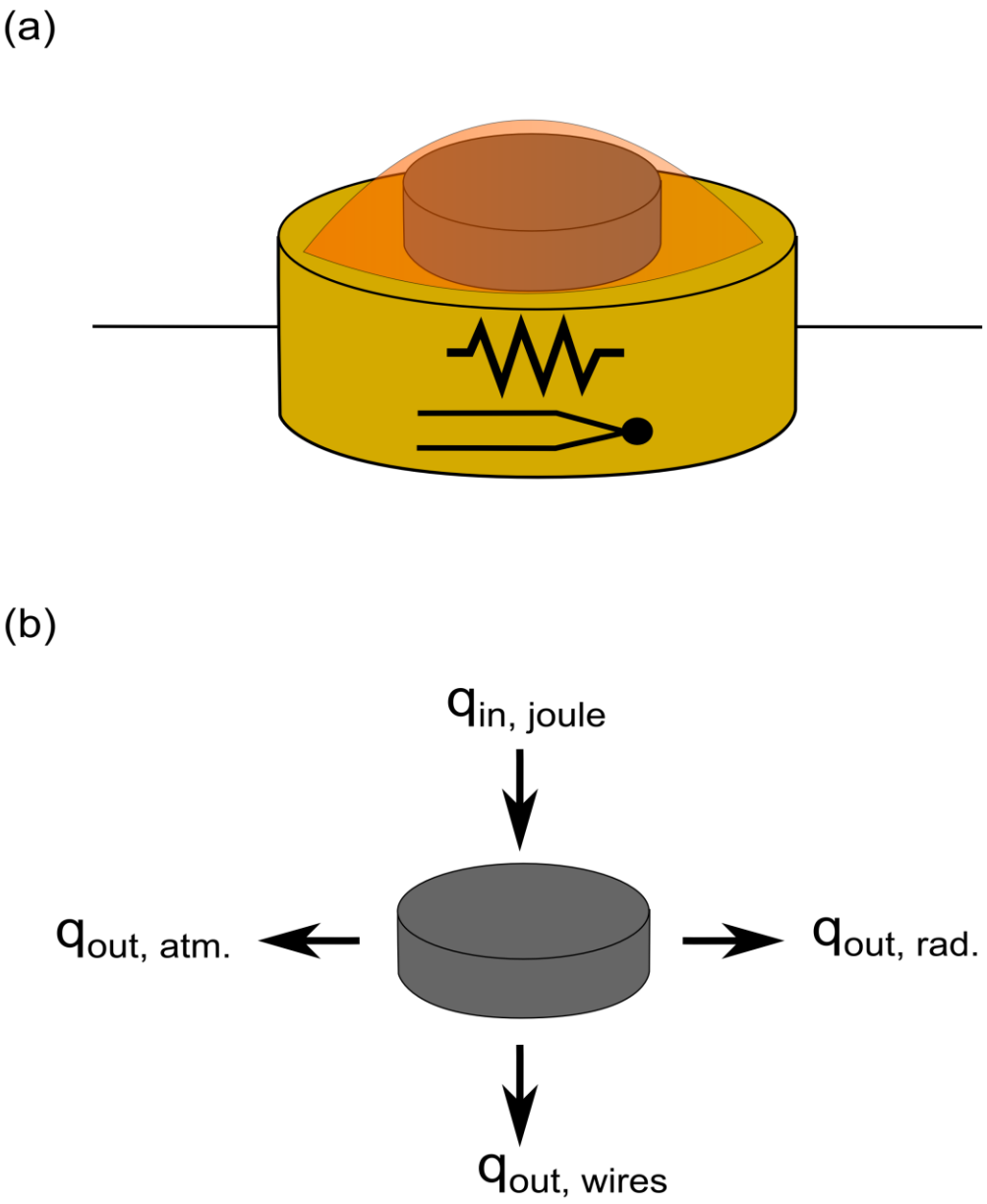


Fig. 1.10 (a) A schematic of a sample stage for measurements of heat capacity by the semi-adiabatic pulse technique **(b)** The heat balance for a sample during a semi-adiabatic pulse tecnique measurement.

1.6 Charge Separation in Photovoltaics

1.6.1 Photovoltaics Based on p-n Homojunctions

Photovoltaic (PV) devices are one of the most promising technologies for achieving cost-competitive, low-carbon energy production and are an extremely active area of research. Traditional PV devices are based on the p-n junction, a change in dopant type or concentration within a semiconductor such that one side is *p-type* (i.e. holes are the majority carrier) and the other side is *n-type* (i.e. electrons are the majority carrier). The net diffusion of carriers across this region creates a *depletion region* in which no carriers are present and also creates an internal electric field. Electrons and holes are generated by the absorption of photons and when these diffuse near the depletion region, they are separated by drift due to the electric field. These carriers accumulate on opposite sides of the junction where they can provide current for an electrical circuit.

The characteristic equation of solar cells provides a good description of the *I-V* curve of a solar cell:

$$i_{out} = i_L - i_0\left(e^{\frac{q(V+i_{out}R_{series})}{nkT}} - 1\right) - \frac{V + i_{out}R_{series}}{R_{SH}}. \qquad (\mathbf{1.14})$$

The currents and resistances contained in Eqn. 1.14 are those shown in the circuit diagram in Fig. 1.10a. i_{out} is the current measured at the terminals of the device and V is the voltage. i_L is the light current, the number of free carriers separated before any losses are taken into consideration. i_0 is the *reverse saturation* current which describes the diffusion of carriers across the depletion region in the direction opposite to the electric field (the term containing i_0 describes the diode current, i_D, in Fig. 1.10a.). R_{series} is the internal resistance of the solar cell and R_{SH} is a resistance

value that describes losses due to parasitic current pathways within the device. *q, k,* and *T* are the fundamental charge, the Boltzmann, and the temperature of the cell, respectively.

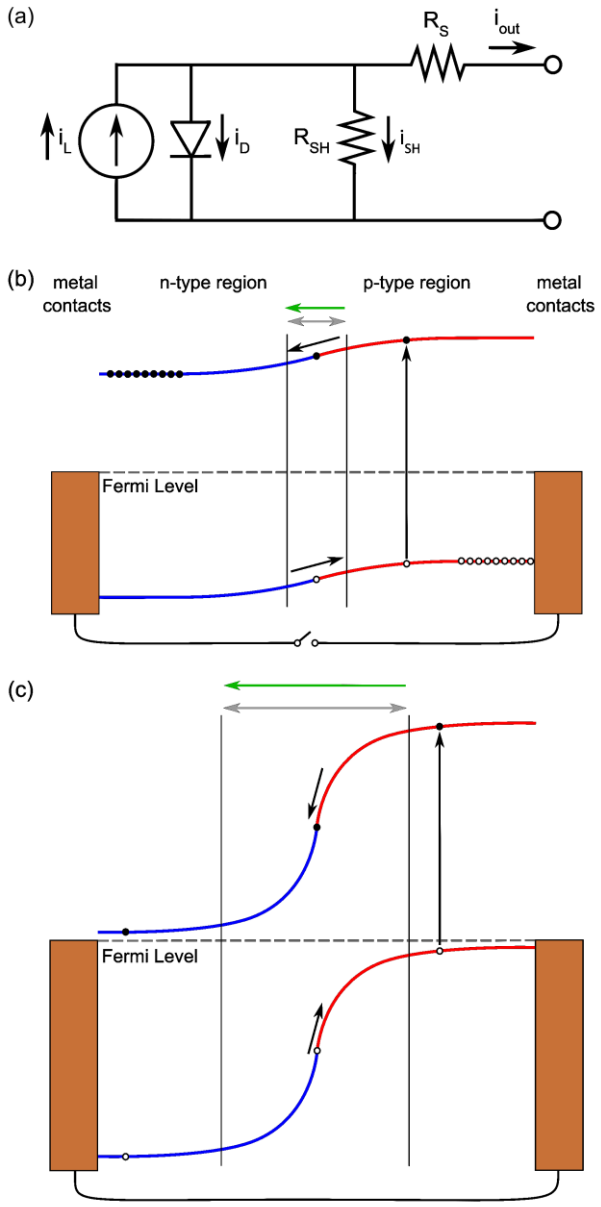


Figure 1.11 (a) The equivalent circuit of a solar cell. **(b)** Charge separation in a PV devices operating near V_{OC}. The green arrow represents the direction and strength of the electric field and the gray arrow indicates the depletion region. **(c)** Charge separation in the same device near I_{SC}

Two experimentally determined values which are often used to describe a PV cell are the *open-circuit voltage,* V_{OC}, and the *short-circuit current*, i_{SC}. As the load across the terminals is increased to infinity, i_{out} decreases and free electrons and holes accumulate in the n-type and p-type regions, respectively. This accumulation decreases the width of the depletion region and the strength of the electric field. This condition is a special case of forward bias for a p-n device in which the bias is provided by the separation of optically generated carriers. The voltage across the terminals in this limit is V_{OC} and is given by

$$V_{OC} \approx \frac{nkT}{q} ln\left(\frac{i_l}{i_0} - 1\right). \qquad (\mathbf{1.15})$$

The p-n junction of a solar cell near the limit of V_{OC} is shown in Fig. 1.11b. As the resistance across the terminals is decreased to zero, the second and third terms in the right-hand side of Eqn.1.14 will also decrease to zero. In this limit the current across the terminals is i_{SC} and is equal to i_L. As the current approaches i_{SC}, the depletion region widens and the electric field increases. Fig. 1.11c depicts a PV device near i_{SC}.

Two important measures of the performance of a PV device are the *energy conversion efficiency*, η, and the *quantum efficiency*. The energy conversion efficiency is given by

$$\eta = \frac{P_m}{I \cdot A}, \qquad (\mathbf{1.16})$$

where P_m is the maximum power collected from the device, I is the intensity of the incident light, and A is area of the device being irradiated. Quantum efficiency is the ratio of the number of photons irradiating the device to current exiting the device at

the terminals (usually in terms of number of electrons, rather than Amperes). Quantum efficiency is usually specified as either the *external quantum efficiency* (EQE), in which case all the photons arriving at the device are considered, or the *internal quantum efficiency* (IQE), in which only the photons absorbed by the device are counted.

1.6.2 Nano-structured Photovoltaics

Early PV devices were mostly based on indirect-bandgap semiconductors such as Si and Ge in which dopants are introduced in order to produce a p-n junction. Later a large number of *thin-film* devices were developed, that are based on a *heterojunction* between two different direct-bandgap semiconductors. The higher absorption coefficient of the direct-bandgap materials allows thinner absorber layers to be used, often thinner than one micron. A tremendous amount of research has also been performed on organic photovoltaics (OPVs). Most OPV designs are based on a combination of two organic semiconductors in which charge separation is driven by an offset in energy between the LUMOs (for electrons) and HOMOs (for holes) of the two materials. Because charge separation requires the diffusion of electrically neutral excitons to the interface, efficient OPV designs rely on a *bulk heterojunction* of two semiconducting polymers which are intimately co-mingled on the nano-scale.[29,30]

Dye-sensitized solar cells (DSSCs), also called *Grätzel cells*, are based on coordination complex dyes which are deposited on TiO_2 nanostructures.[31] The high surface area of the nanostructure increases the loading of the dye far beyond that achieved with a planar structure. Charge separation occurs when an electron is

injected from an excited state of the dye into the nanostructures. The injected electrons diffuse to an electrode where they are collected and the dye is regenerated by an iodine/iodide liquid electrolyte.

Elaboration on the DSSC concept led to the development of perovskite solar cells. *Perovskite* refers to an enormous class of materials with the perovskite crystal structure or closely related crystal structures. The semiconducting perovskites that are commonly used in experimental PV devices have the formula AMX_3 where A is an organic cation, M is Pb or Sn, and X is a Br, Cl, I or a combination thereof. Such perovskites were originally used to replace the dye/sensitizer in liquid electrolyte-based DSSCs.[32,33] Later it was discovered that the perovskite itself could function as both an absorber and a hole-transport material, eliminating the need for the electrolyte.[34,35] The micro- and nanostructure of the perovskite and device performance are highly sensitive to the preparation methods used. Understanding the relationship between nano-structure and charge separation in these devices is an active area of research.

1.6.3 Factors Limiting efficiency

The theoretical upper limit of the fraction of the energy contained in sunlight that can be converted to electrical energy by a single-junction PV device, the Shockley-Queisser limit, is 33.7%.[36] By far the largest source of loss contained in this limit is *spectrum losses*. In an ideal solar cell, photons with energy less than the bandgap do not contribute to the photocurrent. When photons with energy greater than the bandgap are absorbed, the excess energy is rapidly converted to heat as electrons relax to the bottom of the conduction band. The spectrum of sunlight is

approximately a blackbody spectrum with regions of diminished intensity due to atmospheric absorption that varies with altitude, weather, and angle of incidence. The ideal bandgap for minimizing spectrum losses using the common reference spectrum, known as A.M. 1.5, is approximately 1.1 eV. Conveniently, this is also the bandgap of the most commonly used material for PV cells, silicon. Other sources of loss contained in the Shockley-Queisser limit are due to blackbody emission from the device and direct recombination.

In real devices, losses not accounted for in the theoretical limit may be substantial. All real devices contain current pathways between the p- and n-doped regions, which are described in Eqn. 1.14 by the shunt resistance, R_{SH}. Additionally, all real materials contain defects that may act as recombination centers. In silicon PV devices[37] and GaAs/InGaAs quantum well PV devices[38] the saturation of recombination centers with increasing light intensity has been shown to increase efficiency. This effect is part of the rationale for the development of *concentrated photovoltaics* in which optical elements are used to focus sunlight onto a small-area photovoltaic device in order to increase the quantum efficiency. In heterojunction devices and devices employing transport materials into which the injection of carriers is energetically favorable, any energetically favorable carrier injection will reduce V_{OC} by an amount equal to the energy offset between the bands of the materials.

1.6.4 Efficiency beyond the Shockley-Quiesser Limit

Several approaches to increasing the efficiency of PV devices beyond the Shockley-Quiesser limit have been demonstrated in the laboratory. One approach involves using optical elements to disperse sunlight into spectral bands that impinge

on cells that are optimized for those bands.[39] Alternately, multiple-junction devices contain a series of absorbers of progressively smaller bandgap such that incident photons penetrate to the layer at which they can be converted most efficiently.[40] Both of these approaches add considerable complexity to devices.

Another, potentially simpler, approach is the development of meta-materials that can generate separate multiple charges upon absorption of a single high-energy photon, a process known as multiple-exciton generation (MEG). To date, efficient MEG has been observed mostly in nanocrystals. In some materials, such as PbSe quantum dots, it may arise from impact ionization.[41,42] Impact ionization is a process in which electrons in the conduction band with an energy at least twice that of the bandgap scatter from electrons in the valence band, imparting enough energy to promote them to the conduction band. Other theories of MEG in nanocrystals involve quantum mechanical effects such as quantum superposition of single- and multi-exciton states[43] or through the formation of a virtual bi-exciton.[44] In both of these theories, MEG is dependent on the narrowing of bands that occurs in very small particles.

1.7 Conclusions

The technology of phoconducting nanomaterials and nano-structured devices has advanced rapidly in recent years, but much of the advancement has proceeded through trial-and-error. In order to accelerate advancement in this field, it would be useful to develop methods for studying the mechanisms of nano-scale charge separation. Existing crystallographic methods such as PDF can enable researchers to

spacially resolve nano-scale charge separation both in real space and reciprocal space and identify the crystallographic sites, structures, or interfaces involved in charge separation. Improved methods for bulk measurements of light-induced changes in physical properties are also important, especially if they involve the modulation of a variable that isolates different contributions to net charge separation. This work represents progress towards improving and combining these methods so that mechanisms of charge separation can be identified in materials and devices.

1.8 References

(1) Kutty, T. R. N.; Gomathi Devi, L.; Murugaraj, P. *Mater. Res. Bull.* **1986**, *21*, 1093–1102.

(2) Fink, J.; Schierle, E.; Weschke, E.; Geck, J.; Hawthorn, D.; Soltwisch, V.; Wadati, H.; Wu, H.-H.; Dürr, H. A.; Wizent, N.; Büchner, B.; Sawatzky, G. A. *Phys. Rev. B* **2009**, *79*, 100502.

(3) Radaelli, P. G.; Cox, D. E.; Marezio, M.; Cheong, S.-W.; Schiffer, P. E.; Ramirez, A. P. *Phys. Rev. Lett.* **1995**, *75*, 4488–4491.

(4) Chen, C. H.; Fleming, R. M. *Solid State Commun.* **1983**, *48*, 777–779.

(5) Verwey, E. J. W. *Nature* **1939**, *144*, 327–328.

(6) Senn, M. S.; Wright, J. P.; Attfield, J. P. *Nature* **2011**, *481*, 173–176.

(7) Jirák, Z.; Krupička, S.; Šimša, Z.; Dlouhá, M.; Vratislav, S. *J. Magn. Magn. Mater.* **1985**, *53*, 153–166.

(8) Tranquada, J. M.; Buttrey, D. J.; Lorenzo, J. E.; Sachan, V. *Phys. B Phys. Condens. Matter* **1995**, *213-214* (C), 69–71.

(9) Zaanen, J.; Gunnarsson, O. *Phys. Rev. B* **1989**, *40*, 7391–7394.

(10) Peierls, R. *Quantum Theory of Solids*; Oxford University Press: London, 1955.

(11) Rouxel, J.; Meerschaut, A.; Gressier, P. *Berichte der Bunsengesellschaft für Phys. Chemie* **1983**, *87*, 263–268.

(12) Wilson, J. A.; Di Salvo, F. J.; Mahajan, S. *Adv. Phys.* **1975**, *24*, 117–201.

(13) Ruani, G.; Pal, A.; Taliani, C.; Zamboni, R.; Wei, X.; Chen, L.; Vardeny, Z. V. *Synth. Met.* **1991**, *43*, 3977–3980.

(14) Andersson, G.; Parck, C.; Ulfvarson, U.; Stenhagen, E.; Thorell, B. Studies on Vanadium Oxides. II. The Crystal Structure of Vanadium Dioxide. *Acta Chemica Scandinavica*, 1956, *10*, 623–628.

(15) Seidel, A.; Marianetti, C.; Chou, F. C.; Ceder, G.; Lee, P. *Phys. Rev. B* **2002**, *67*, 020405.

(16) Day, P. *Low-Dimensional Cooperative Phenomena*; Keller, H. J., Ed.; Springer US: Boston, MA, 1975.

(17) Ong, N. P.; Monceau, P. *Phys. Rev. B* **1977**, *16*, 3443.

(18) Raičević, I.; Jaroszyński, J.; Popović, D.; Panagopoulos, C.; Sasagawa, T. *Phys. Rev. Lett.* **2008**, *101*, 177004.

(19) Bragg, W. H.; Bragg, W. L. *Proc. R. Soc. A Math. Phys. Eng. Sci.* **1913**, *88*, 428–438.

(20) Rietveld, H. M. *J. Appl. Crystallogr.* **1969**, *2*, 65–71.

(21) Bakulin, A. A.; Rao, A.; Pavelyev, V. G.; van Loosdrecht, P. H. M.; Pshenichnikov, M. S.; Niedzialek, D.; Cornil, J.; Beljonne, D.; Friend, R. H. *Science* **2012**, *335*, 1340–1344.

(22) Jailaubekov, A. E.; Willard, A. P.; Tritsch, J. R.; Chan, W.-L.; Sai, N.; Gearba, R.; Kaake, L. G.; Williams, K. J.; Leung, K.; Rossky, P. J.; Zhu, X.-Y. *Nat. Mater.* **2013**, *12*, 66–73.

(23) Grancini, G.; Maiuri, M.; Fazzi, D.; Petrozza, A.; Egelhaaf, H.-J.; Brida, D.; Cerullo, G.; Lanzani, G. *Nat. Mater.* **2013**, *12*, 29–33.

(24) Ogawa, N.; Shiraga, A.; Kondo, R.; Kagoshima, S.; Miyano, K. *Phys. Rev. Lett.* **2001**, *87*, 256401.

(25) Gaál, R.; Donovan, S.; Sörlei, Z.; Mihály, G. *Phys. Rev. Lett.* **1992**, *69*, 1244–1247.

(26) Fausti, D.; Tobey, R. I.; Dean, N.; Kaiser, S.; Dienst, A.; Hoffmann, M. C.; Pyon, S.; Takayama, T.; Takagi, H.; Cavalleri, A. *Science* **2011**, *331*, 189–191.

(27) Stockmann, F. *Naturwissenschaften* **1952**, *39*, 246-254.

(28) Testardi, L. *Phys. Rev. B* **1971**, *4*, 2189–2196.

(29) Yu, G.; Gao, J.; Hummelen, J. C.; Wudl, F.; Heeger, a. J. *Science* **1995**, *270*, 1789–1791.

(30) Halls, J. J. M.; Walsh, C. A.; Greenham, N. C.; Marseglia, E. A.; Friend, R. H.; Moratti, S. C.; Holmes, A. B. *Nature* **1995**, *376*, 498–500.

(31) O'Regan, B.; Grätzel, M. *Nature* **1991**, *353*, 737–740.

(32) Kojima, A.; Teshima, K.; Shirai, Y.; Miyasaka, T. *J. Am. Chem. Soc.* **2009**, *131*, 6050–6051.

(33) Im, J.-H.; Lee, C.-R.; Lee, J.-W.; Park, S.-W.; Park, N.-G. *Nanoscale* **2011**, *3*, 4088–4093.

(34) Lee, M. M.; Teuscher, J.; Miyasaka, T.; Murakami, T. N.; Snaith, H. J. *Science* **2012**, *338*, 643–647.

(35) Kim, H.-S.; Lee, C.-R.; Im, J.-H.; Lee, K.-B.; Moehl, T.; Marchioro, A.; Moon, S.-J.; Humphry-Baker, R.; Yum, J.-H.; Moser, J. E.; Grätzel, M.; Park, N.-G. *Sci. Rep.* **2012**, *2*, 591.

(36) Shockley, W.; Queisser, H. J. *J. Appl. Phys.* **1961**, *32*, 510.

(37) Dalal, V. L.; Moore, A. R. *J. Appl. Phys.* **1977**, *48*, 1244.

(38) Yang, M.; Yamaguchi, M. *Sol. Energy Mater. Sol. Cells* **2000**, *60*, 19–26.

(39) Warmann, E. C.; Eisler, C.; Kosten, E.; Escarra, M.; Atwater, H. a. *Conf. Rec. IEEE Photovolt. Spec. Conf.* **2013**, 1922–1925.

(40) Yamaguchi, M.; Takamoto, T.; Araki, K.; Ekins-Daukes, N. *Sol. Energy* **2005**, *79*, 78–85.

(41) Franceschetti, A.; An, J. M.; Zunger, A. *Nano Lett.* **2006**, *6*, 2191–2195.

(42) Beard, M. C.; Midgett, A. G.; Hanna, M. C.; Luther, J. M.; Hughes, B. K.; Nozik, A. J. *Nano Lett.* **2010**, *10*, 3019–3027.

(43) Shabaev, A.; Efros, A. L.; Nozik, A. J. *Nano Lett.* **2006**, *6*, 2856–2863.

(44) Rupasov, V. I.; Klimov, V. I. *Phys. Rev. B* **2007**, *76*, 125321.

Chapter 2: Dynamic charge disproportionation in the 1D chain material PdTeI

This work was co-written with the following authors and is published under the following citation:

Patrick Cottingham, David C. Miller, John P. Sheckelton, James R. Neilson, Mikhail Feygenson, and Tyrel M. McQueen

J. Mat. Chem. C. **2014**, *2*, 3238-3246.

2.1 Introduction

The compound PdTeI features quasi-1D chains of corner-sharing $PdTe_4I_2$ octahedra along the *c*- axis (Fig. 2.1a). These chains form bundles of four chains in which each octahedron is edge-sharing with four neighboring octahedra within its own bundle, and edge-sharing with one octahedron in an adjacent bundle. Groups of four bundles enclose large channels with a radius of 4.3 Å in which all the channel-facing atoms are iodine ions (Fig. 2.1b).

PdTeI was originally characterized using single crystal x-ray diffraction and assigned the $P4_2/mmc$ space group with lattice parameters of a = 7.808(2) Å and c = 5.660(1) Å.[1] This structure is intriguing given that, assuming oxidation states of 2- and 1- for tellurium and iodine, as bond valence sums suggest, the formal oxidation state of Pd in this material is 3+. This is a highly unusual oxidation state for palladium, which instead tends to disproportionate into Pd^{2+} and Pd^{4+}. Organometallic complexes containing Pd^{3+} have only recently been discovered[2] and solid-state inorganic materials featuring Pd^{3+} are rare.[3,4]

Further, charge and orbital orders are well-known in isoelectronic Ni^{3+} and Pt^{3+} compounds.[5,6] In some cases, the charge rearrangement is observed by standard crystallographic techniques. However in other cases, the charge or orbital order has been found to be restricted to short length scales.[7]

In the published structure of PdTeI, all Pd atoms are in identical coordination environments, indicating the absence of disproportionation, even though band structure calculations have predicted that PdTeI should undergo a *c*- axis doubling distortion.[8] The original report of the structure of PdTeI also reported an anomalously large value of the thermal parameter U_{33} for iodine, suggesting that a structural distortion allowing for charge disproportionation could involve splitting of the I site.

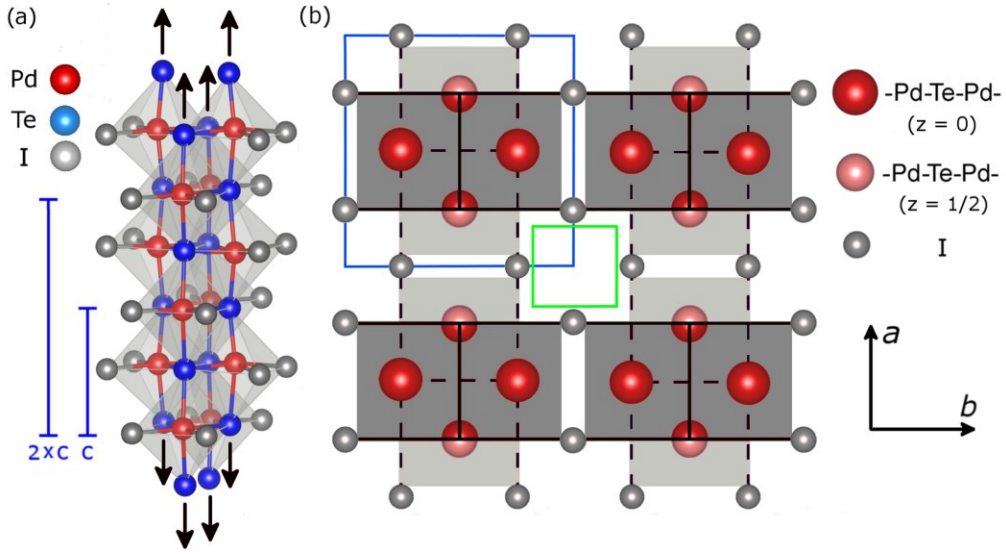


Fig. 2.1 (a) Illustration of a bundle of four 1D -Pd-Te-Pd- chains. The crystal structure of PdTeI is formed by each octahedron on a face of the bundle sharing an edge with an octahedron of an adjacent bundle. **(b)** The structure of PdTeI as viewed in the [00l] direction. Te atoms are located above and below Pd atoms. The blue line indicates a single bundle of four 1D chains. The green line indicates a channel enclosed by I atoms.

In this work, high-resolution crystallographic techniques and materials properties measurements were used to probe the electronic configuration in PdTeI. Our results show that there is *dynamic* charge disproportionation of Pd^{3+}, even at room temperature. As the temperature is lowered from room temperature, the charge separation becomes highly correlated, followed by static charge order below $T_{CDW} = 50$ K. To our knowledge, this is the first direct observation of a dynamic to static charge order transition in a 4*d* transition metal material.

2.2 Experimental

2.2.1 Synthesis of PdTeI

Polycrystalline samples of PdTeI were prepared by a modification of the method of Seo et al.[8] 140 mg of a mixture of Pd (Alfa Aesar 99.95%), Te (Alfa Aesar 99.9%), and KI (Alfa Aesar 99.999%) in a 2:2:1 molar ratio were combined and intimately ground with a mortar and pestle. This mixture was pressed into a pellet and combined with 120 mg (0.47 mmol) of I_2 (Sigma-Aldrich 99.8%) and 2 mL of 57% (wt. %, Sigma-Aldrich) aqueous HI (overall Pd:Te:I ratio 2:2:37) and sealed inside a silica tube of 10 mm inner diameter. The sample was placed in a furnace and heated at 20 °C/hr to 300 °C, held at 300 °C for 72 hours and then cooled at 20 °C/hr. The tubes were subsequently opened and the products were washed over a filter paper with deionized water. Initial characterization was performed by powder x-ray diffraction (PXRD) using a Bruker D8 diffractometer and copper *Kα* radiation.

2.2.2 Physical Properties Measurements

Physical properties measurements were carried out using a Quantum Design Physical Properties Measurement System (PPMS). Temperature-dependent resistivity measurements were collected on a polycrystalline bar of sample in the four-probe configuration. Magnetic susceptibility and heat capacity measurements were carried out on a polycrystalline pellet and a powdered sample, respectively.

2.2.3 X-ray Scattering Measurements

High-resolution synchrotron x-ray diffraction (SXRD) data were collected at T = 300, 220, 180, 140, 80, 50, 30, 12, and 6.7 K on beam-line 11-BM at the Advanced Photon Source (APS) at Argonne National Laboratory with a photon wavelength of λ = 0.4131(1) Å and step size of 0.001° 2θ. Rietveld-refinements to the SXRD data were made using the GSAS/EXPGUI package.[9,10] Because of extreme asymmetry of some Bragg reflections in the SXRD data at low temperatures, peak shapes could not be accurately accounted for in a Rietveld refinement using the original binning. Thus, data were re-binned with a coarser spacing of 0.01° 2θ to allow for Rietveld refinement of peak intensities. X-ray total scattering data were collected at room temperature using the 11-ID-B beamline at the Advanced Photon Source with a photon wavelength of λ = 0.1370(1) Å. A reduced scattering structure function, S(Q), with the appropriate corrections for multiple scattering, sample absorption, x-ray polarization, and Compton scattering was obtained from the data using the program PDFgetX2.[11] A pair distribution function (PDF), G(r), was obtained by direct Fourier transformation of S(Q) with a Q_{max} = 25 Å^{-1}. X-ray PDFs were analyzed using the program PDFGUI.[12]

2.2.4 Neutron Scattering Measurements

Powder neutron diffraction data sets at T = 300, 220, 180, 140, 100, 80, 70, 50, 30 and 12 K were collected at the Spallation Neutron Source (SNS) at the Oak Ridge National Laboratory using the POWGEN diffractometer (BL-11A) and analyzed with the Rietveld method using GSAS/EXPGUI. Neutron total scattering data were collected at T = 300, 100, 60, 50, 45, 30, and 7 K using the Nanoscale-Ordered Materials Diffractometer (NOMAD) instrument at the SNS. A reduced scattering structure function, S(Q), with the appropriate corrections for scattering by the sample holder and vanadium container was obtained from the data using the software developed for the NOMAD instrument. A pair distribution function (PDF), G(r), was obtained by direct Fourier transformation of S(Q) with a Q_{max} = 25 Å^{-1}. Neutron PDFs were analyzed using the program PDFGUI. Energy-dispersive X-ray Spectroscopy (EDX) was performed using a JEOL 6700F Scanning Electron Microscope with EDX analyser and an electron energy of 10 keV.

2.3 Evidence of Ordering from Physical Properties

Temperature-dependent resistivity measured on an as-synthesized polycrystalline pellet is shown in Fig. 2.2. There are two significant changes on cooling. The first is a clear discontinuity at T_{CDW} = 50 K, indicative of a phase transition. The shape and magnitude of the upturn is similar to that found in well-known CDW materials[13,14], suggesting that PdTeI undergoes a CDW transition at T_{CDW} = 50 K. Further, from T = 50 K to $T \approx$ 120 K, there is a broad maximum in the resistivity, suggesting that CDW

correlations develop prior to the phase transition. There is a hysteresis over that same region; a similar thermal hysteresis in resistivity has been observed on CDW formation in $Lu_5Rh_4Si_{10}$ and $Nb_{10}Se_{40}I_3$.[15,16] Measurements of magnetic susceptibility vs. temperature (shown in Fig. 2.2 inset) show only a small, temperature-independent paramagnetic signal with no discontinuities or changes in slope that might indicate the presence of magnetic order or changes to a magnetic structure.

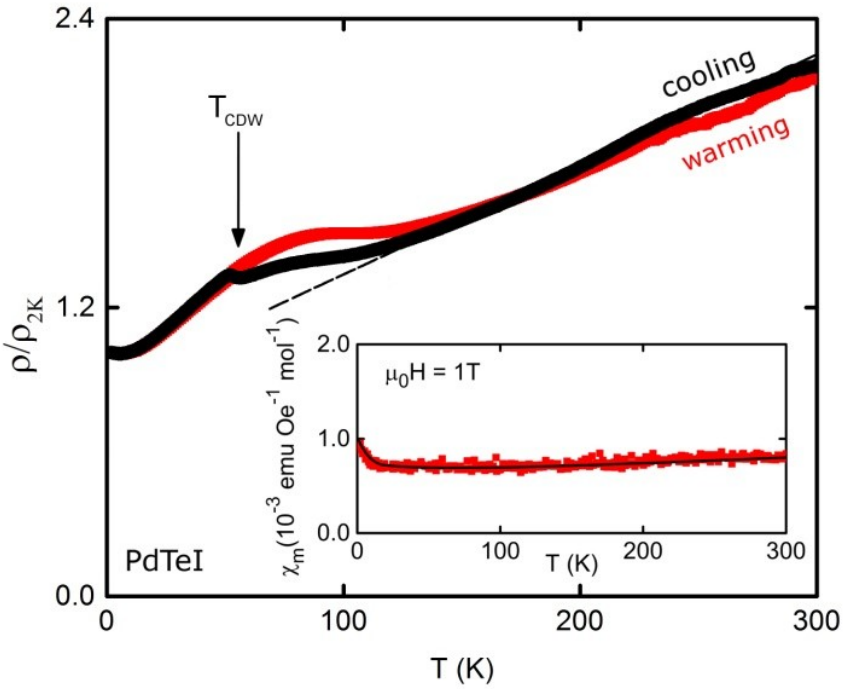


Fig. 2.2 The normalized resistivity of PdTeI. The dashed line is an extrapolation to $T = 80$ K of the resistivity above $T \approx 120$K. The resistivity of sample at $T = 2$ K is $\rho = 0.8$ mΩ·cm, with the dominant contribution to the resistivity due to grain boundaries. (inset) The magnetic susceptibility of PdTeI, which displays paramagnetic behavior.

Specific heat measurements show no peaks or indication of large quantities of entropy loss at T_{CDW} in the region of the maximum of the resistivity. This implies that

either the phase transition is first order (the semi-adiabatic pulse technique is not sensitive to first order transitions), or that the entropy involved is small. The former is unlikely, as neither inspection of the individual temperature-versus-time heat pulse traces (Fig. 2.3) nor temperature-dependent structural data (Fig. 2.7) show discontinuities on heating or cooling that might indicate a first order phase transition.

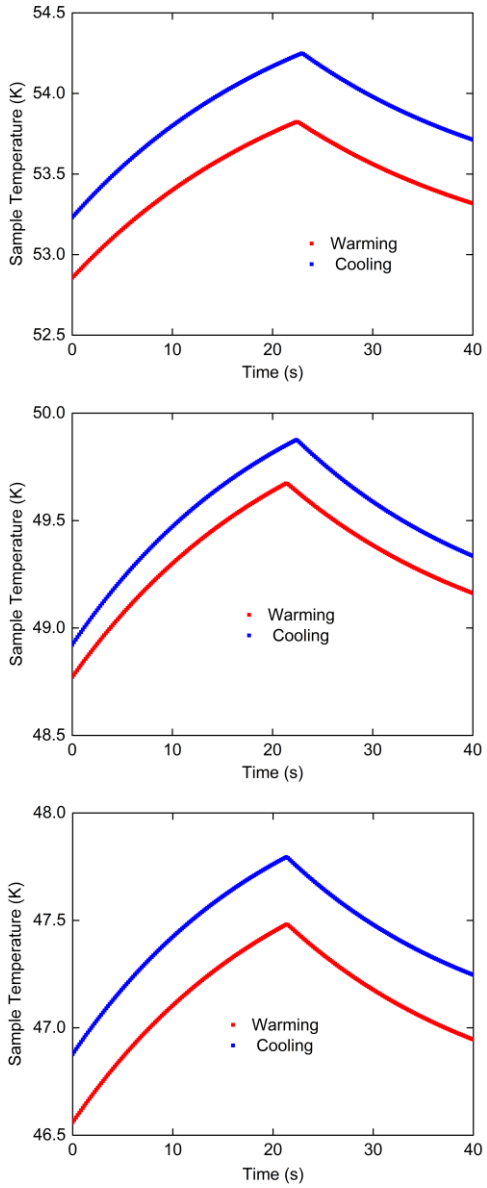


Fig. 2.3 Individual heat capacity traces obtained using the semi-adiabatic pulse technique in the vicinity of T_{CDW} = 50 K.

A plot of C_P/T^3 versus ln(T) can be used to isolate various phonon contributions to the specific heat. Debye modes, which describe strongly dispersing phonon branches such as acoustic waves, plateau at low temperature on such a plot, while Einstein modes, which describe low dispersion modes, such as localized optic vibrations, appear as local maxima. Such a plot is shown for PdTeI in Fig. 2.4. There is a sharp peak in the normalized heat capacity around T = 9.2 K, indicating the presence of at least one strong Einstein mode. The best fit to the data with the fewest parameters was obtained using a model including an electronic contribution and a lattice contribution comprising two Debye modes and one Einstein mode. While this model captures the essential features of the data, more complex models for the low-temperature heat capacity cannot be ruled out. Debye modes have a characteristic cutoff frequency (v_m), conventionally reported as a Debye temperature, $\theta_D = hv_m/k_B$ were h is Planck's constant and k_B is Boltzmann's constant. The characteristic frequency of an Einstein mode (v) is conventionally reported as an Einstein temperature using the same conversion relation. For our model these temperatures are: θ_{D1} = 299(3) K, θ_{D2} = 116(5) K, and θ_E = 44.0(1) K. The Einstein temperature (θ_E) is near to the phase transition temperature in the resistivity and suggests that the two are related.

The above analysis of phonon contributions is consistent with the Debye T^3 behavior found in the low temperature limit. A fit of the low temperature heat capacity to $C_P/T = \gamma + \beta T^2$ (Fig. 2.4 inset) gives values of γ = 2.85(2) mJ K^{-2} mol f.u.$^{-1}$ and β = 1.55(5) mJ K^{-4} mol f.u.$^{-1}$ for the electronic and lattice contributions respectively. This value of β corresponds to an effective Debye temperature of θ_D = 143(1) K, which is

between the two Debye temperatures found from the analytical phonon fit to all the specific heat data, as expected.

PdTeI has previously been reported to be either a small-bandgap semiconductor or metallic. The presence of a sizeable electronic contribution to the low temperature specific heat implies a significant carrier density in the material. Assuming a nearly-free electron gas model, a free carrier concentration can be calculated from γ using the equation:

$$\gamma = 0.706\, Z \left(\frac{r_s}{a_0}\right)^2 \times\ 10^{-4}\, J\, mol^{-1} K^{-2} \qquad \textbf{(2.1)}$$

where Z is the valence, assumed to be $Z = 1$, a_0 is the Bohr radius, and r_s is the radius of a sphere containing the average real-space volume per free electron.[17] The free carrier concentration calculated in this manner is $n = 6.29(3) \times 10^{21}$ carrier/cm^3, in contrast to a predicted value of $n = 2.911(3) \times 10^{21}$ carrier/cm^{21} if each Pd atom contributes one carrier.

Both of these values are below the carrier density at which a Mott metal-to-insulator transition is expected to take place[18]:

$$n^{1/3}\, a_0 \cong 0.2 \qquad \textbf{(2.2)}$$

The larger of the two values for the carrier density can be used in conjunction with Eqn. 2.3 to place a lower bound on the electron mobility in PdTeI at $T = 2$ K. The electron mobility calculated in this manner is $\mu_{2K} = 1.23$ cm^2 V^{-1} s^{-1}.

$$\mu = {}^{1}/_{\rho_{2K} n |e^-|} \qquad \textbf{(2.3)}$$

The electronic contribution to the specific heat can also be used to calculate the Sommerfeld-Wilson ratio[19]:

$$R_{s-w} = \frac{\pi^2 {k_B}^2 \chi_P}{3 {\mu_B}^2 \gamma} \qquad \textbf{(2.4)}$$

where μ_B is the Bohr magneton and χ_P is the Pauli contribution to the magnetic susceptibility taken from a linear fit of the magnetic susceptibility from T = 100 K to T = 300 K. The calculated value of $R_{S\text{-}W}$ = 2.98 indicates that PdTeI is a metal with non-trivial electron-electron correlations.

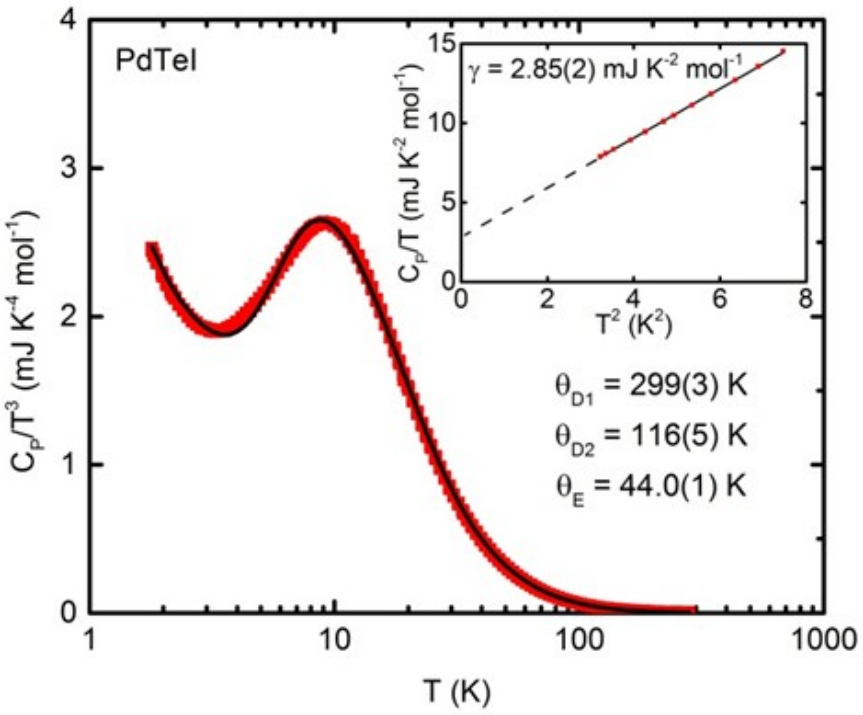


Fig. 2.4 The temperature dependence of the heat capacity of PdTeI. The local maximum in the C_p/T^3 indicates the presence of a weakly dispersing phonon mode. Inset: The intercept of a linear fit of C_p/T vs. T^2 at low temperatures gives a value for the Sommerfeld γ. The dashed line is an extrapolation of the trend to T^2 = 0 K^2.

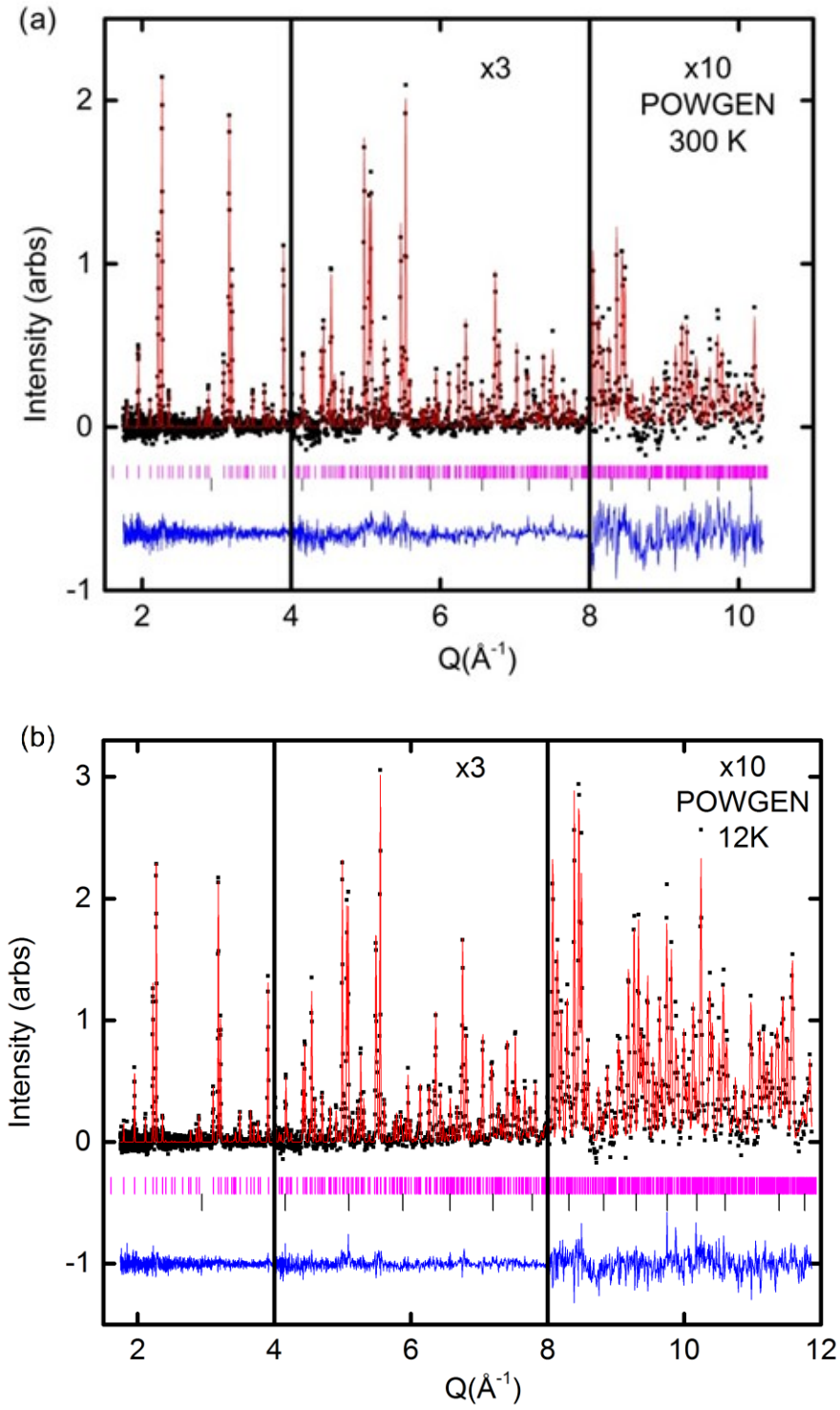
Taken together, these results leave ambiguous whether PdTeI is a semiconductor, a semimetal, or a metal. The Sommerfeld-Wilson ratio suggests that it is a metal, however the carrier density suggests that it is a semiconductor. The measured resistivity

is intermediate between those of semiconductors and metals. The lower bound for the charge carrier mobility is in the metallic range, however the mobility is likely to be higher because the resistivity is dominated by grain boundaries.

2.4 Results from Diffraction Experiments

Medium-resolution powder neutron diffraction (PND) was used to determine the room temperature structure, in order to provide contrast of the Te and I atoms in the structure, which are indistinguishable by x-ray diffraction. Data collected at $T = 300$ K are shown in Fig. 2.5a. All of the observed diffraction peaks can be indexed with a combination of the reported PdTeI unit cell and the vanadium sample container. A Rietveld refinement gave structural parameters in good agreement with the literature, with only minor variations in atom positions and atomic displacement parameters. The final refinement is shown in Fig. 2.5a, and parameters are given in Table 2.1.

PND data collected at lower temperatures down to $T = 12$ K showed no significant changes (Fig. 2.5b), and no additional peaks were observed at any temperature which might arise from an enlarged unit cell due to CDW formation or the presence of magnetic order. To search with higher sensitivity for small peak splittings that might indicate the formation of charge order or a spin-Peierls transition, temperature-dependent high resolution synchrotron powder x-ray diffraction data were collected at the Advanced Photon Source, Beamline 11-BM-B. Results of Rietveld refinements to the room temperature SXRD data using the structure determined by PND, Fig. 2.5c, show that all peaks were accounted for by the PdTeI structure, with a very small 0.55 mol.% of the secondary phase $PdTe_2$.


(a)
x3
x10
POWGEN
300 K
Intensity (arbs)
Q(Å^{-1})
(b)
x3
x10
POWGEN
12K
Intensity (arbs)
Q(Å^{-1})

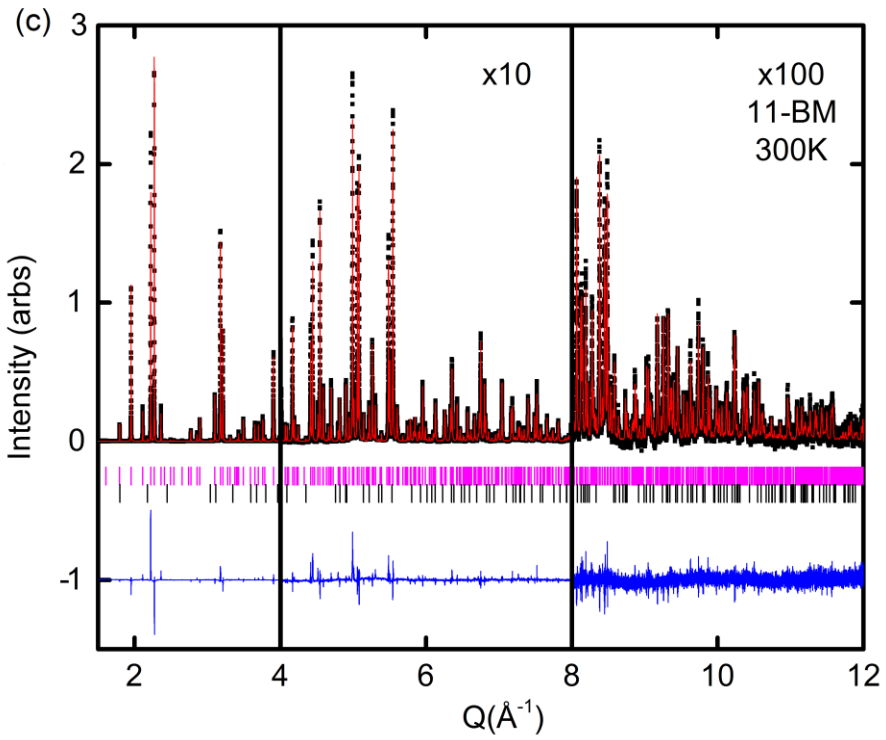


Fig. 2.5 (a) Rietveld analysis of PND data collected at $T = 300$ K with a center wavelength of $\lambda = 1.333$ Å. Black points are diffraction data, the red line is the result of the refinement and the blue line indicates the residual. Pink tickmarks indicate the position of reflections resulting from the PdTeI phase and black tickmarks indicate reflections resulting from the vanadium can. **(b)** PND data at $T = 12$ K. **(c)** Rietveld analysis of SXRD data collected at $T = 300$ K with a wavelength of $\lambda = 0.4131(1)$ Å. Black points are diffraction data, the red line is the result of the refinement and the blue line indicates the residual. Pink tickmarks indicate the position of reflections resulting from the PdTeI phase and black tickmarks indicate reflections resulting from a small amount of $PdTe_2$.

PdTeI		T = 300 K	a = 7.82554(8) Å;	c = 5.66262(9) Å		
Atom	Site	x	y	z	occ.	U_{iso} (Å^2)
Pd	4*l*	0	0.2535(4)	0	1	0.0120(8)
Te	4*j*	0.2171(4)	0	0	1	0.0202(9)
I	4*m*	0.2443(5)	0.5	0	1	0.0244(11)

Table 2.1 Structure parameters of PdTeI at room temperature obtained from Rietveld refinement of powder neutron diffraction data, showing atomic positions, occupancies, and isotropic atomic displacement parameters (U_{iso}). Space group $P4_2/mmc$ (131). The fit statistics are R_p = 0.1336, R_{wp} = 0.0631, and χ^2 = 1.508.

Although additional superstructure peaks or peak splittings were not observed down to T = 6.7 K, there are significant changes in the SXRD data on cooling. Most significantly, there is an asymmetric broadening of the Bragg reflections, as shown in Fig. 2.6. The broadening is *hkl*-dependent, with peaks with larger *l* values broadening more than peaks corresponding to lower *l* values, with peak widths reaching a maximum around T = 50 K. The temperature- dependence suggests a change in the local structure on cooling. The specific *hkl*-dependence of the widths and the asymmetry of the broadening is consistent with the presence of the one dimensional version of stacking faults: that is, the presence of some superstructure that is well-ordered along the *c*-direction, but with a short coherence length in the *a*- and *b*- directions.[20]

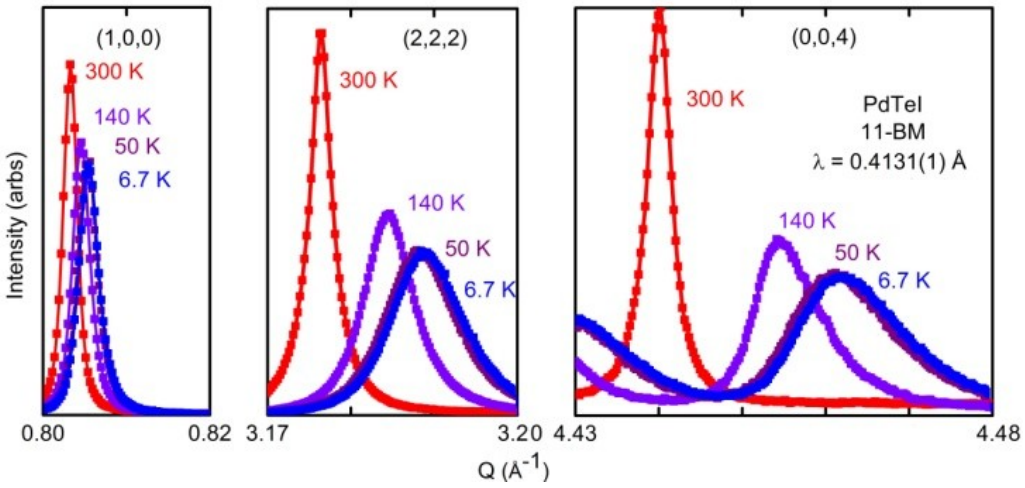


Fig. 2.6 The evolution upon cooling of three representative SXRD Bragg peaks of PdTeI. Lines are provided as a guide to the eye. Different intensity scales are used for each reflection. Data collected on 11-BM-B with an incident wavelength of $\lambda = 0.4131(1)$ Å.

Changes in the lattice parameters determined from fits of the reported structure to SXRD data are consistent with the occurrence of a phase change around $T = 50$ K. The lattice parameter a (Fig. 2.7a) and the ratio c/a (Fig. 2.7b) decrease steadily with decreasing T above T_{CDW} and reach a plateau below T_{CDW}. The full width at half maximum (FWHM) of Bragg diffraction peaks with a non-zero value of the index l increases with decreasing temperature and also reaches a plateau below T_{CDW} (Fig. 2.7c). The FWHM of peaks for which the value of the index l is zero is relatively constant with temperature. Values of the isotropic thermal parameter, U_{iso}, taken from the fits are much larger at all temperatures for Te and I than for Pd (Fig. 2.7d). Most importantly, the temperature dependence of the tellurium atomic displacement parameter is weaker than expected, suggesting that, locally, the Te site split into two compared to the crystallographic average structure.

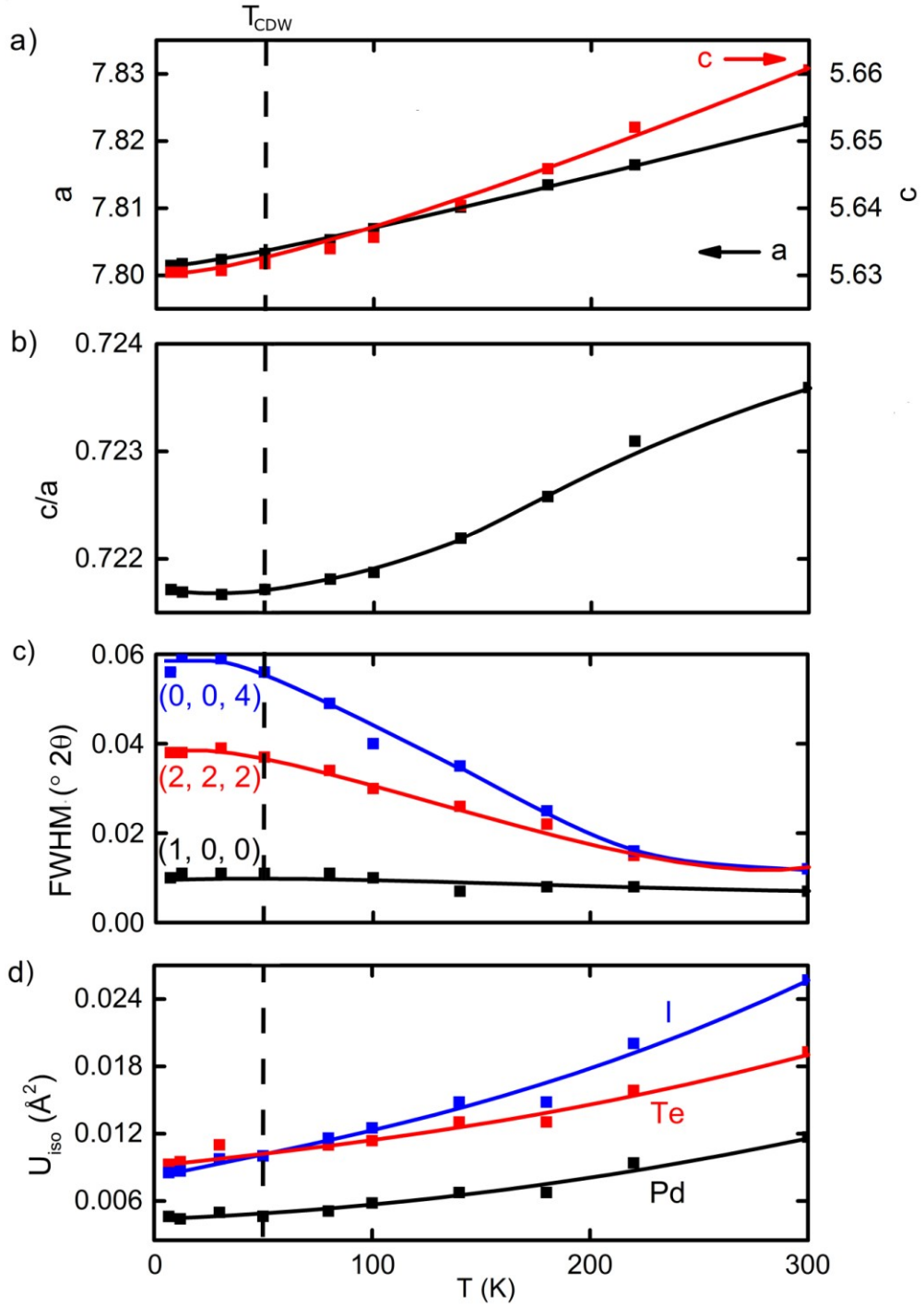


Fig. 2.7 (a), (b) Change with temperature of the lattice parameters *a* and *c* and the ratio *c*/*a* as determined from the SXRD data. **(c)** Change in the full width at half maximum (FWHM) of three representative Bragg peaks. **(d)** The change in U_{iso} determined by the same fits. All lines are guides to the eye.

In order to investigate features of the local structure that might account for this broadening, pair distribution function (PDF) analysis of synchrotron x-ray total scattering data and neutron total scattering data were used. A refinement was performed to the x-ray PDFs from r = 1.5 Å to r = 7 Å. The average structure (Table 2.1) is not sufficient to describe the local structure (Fig. 2.8), particularly the clear atom-atom distance at 5.3 Å which is present in the x-ray PDF but not fit at all by the average structure. This feature is robust to selection of Q_{max} and other PDF extraction parameters (Fig. 2.9).

2.5 Results from Total Scattering Experiments

A number of candidate models were assessed for their ability to fit the features in the PDF which are not explained by the average structure, including: (1) a model in which Te atoms are displaced from the ideal positions in the *ab* plane; (2) a model in which Te atoms are displaced in the *c*- direction; (3) one in which I atoms are displaced in the *c*- direction; (4) and one model in which Pd atoms were displaced along the *c*-direction in order to generate -Pd-Te-Pd- trimers. The best fit to the PDF is obtained by displacing alternating Te atoms along a single chain either up or down along the *c*- axis by the same distance and lowering the symmetry of the local structure to $P4_322$ (R_w = 0.110 vs R_w = 0.121 for the undistorted model). This has the effect of causing the previously indistinguishable elongated Pd octahedra to approach octahedral and square planar coordination, respectively. The introduction of this distortion allows the model to fit the feature in the PDF data around 5.3 Å which is not fit by the average model (Fig. 2.8). This feature corresponds well with the distance between Te and I atoms on

opposite corners of the same Pd polyhedron. Allowing I atoms to displace along the *c*-axis was found to further substantially improve the quality-of-fit (R_w = 0.097).

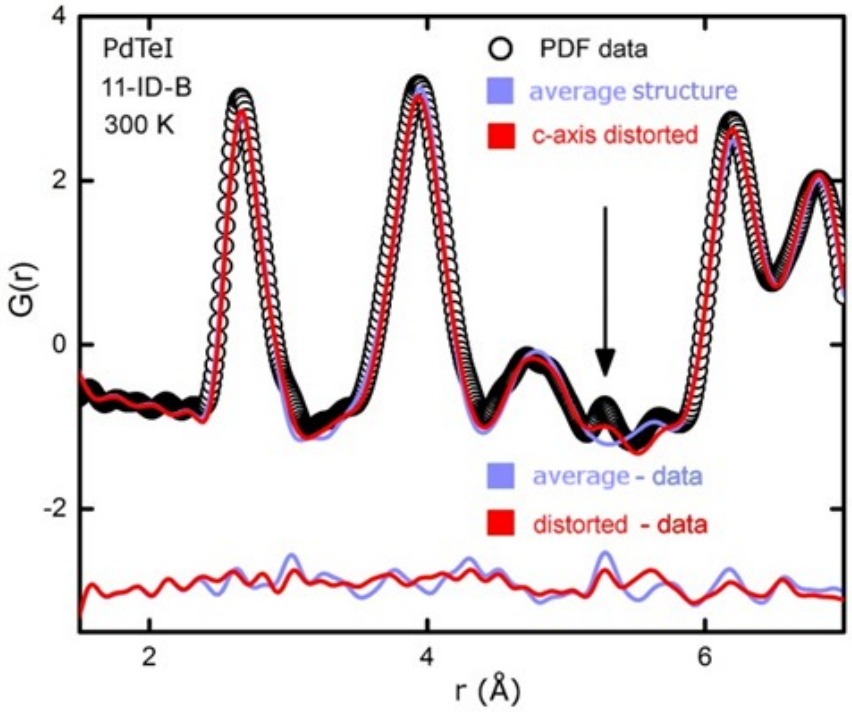


Fig. 2.8 The PDF of PdTeI, extracted from x-ray total scattering data collected at T = 300 K on 11-ID-B with an incident wavelength of λ = 0.1370(1) Å. The light blue curve is a fit to the average model, which clearly is unable to describe the pairwise correlation at 5.3 Å. A distorted model that doubles the *c*- axis and allows for two crystallographically distinct sites describes the data. Residuals are shown below.

The alternating pseudo-square planar and pseudo-octahedral coordination of Pd atoms suggests that these atoms disproportionate towards the 2+ and 4+ oxidation states at room temperature. This disproportionation accounts for the small and temperature-independent magnetic susceptibility of PdTeI given that both d^8 Pd^{2+} in square planar coordination and d^6 Pd^{4+} in low-spin octahedral coordination have no unpaired spins.

Given the relatively small amplitude of the CDW, it is unlikely that the Pd atoms truly exist in integer oxidation states but rather that there is a periodic modulation in the electron density between adjacent Pd atoms. The small amplitude of the distortions also suggests that the electronic ordering in PdTeI is best described as a CDW, rather than a CO phase, because it is inconsistent with the electron hopping that characterizes the CO phases above the Verwey transistion.

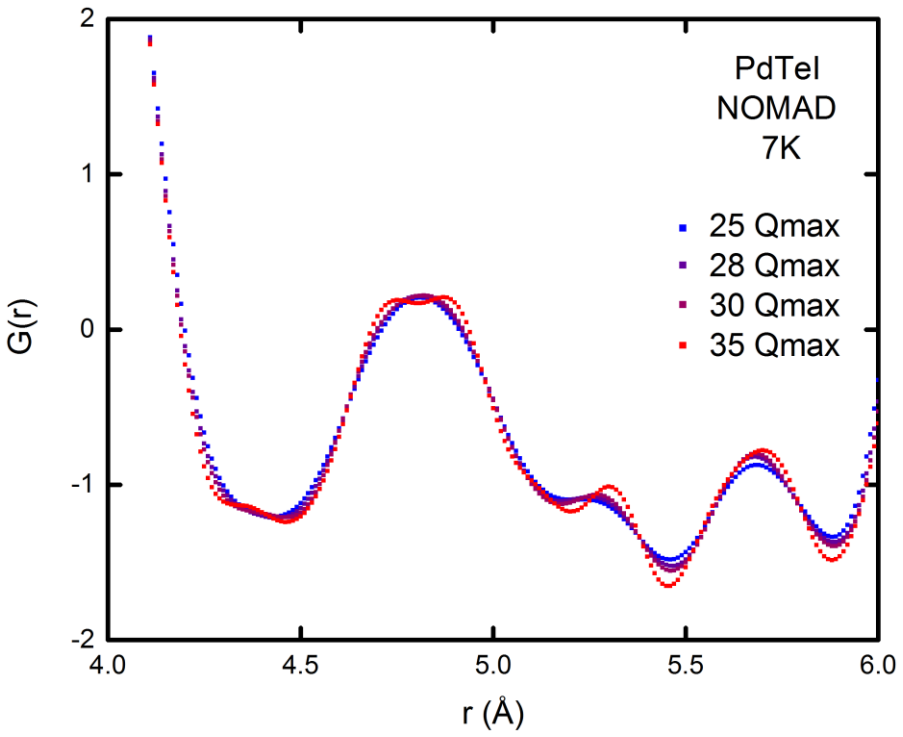


Figure 2.9 The PDF of PdTeI extracted from neutron total scattering data collected at T = 7 K on NOMAD using several values of Q_{max}.

Bundles of *c*- axis chains may also be considered as 1D units of edge-sharing Pd_4 tetrahedra (Fig. 2.10a). Each tetrahedron must contain two Pd atoms in pseudo-octahedral coordination and two in pseudo-square planar coordination in order to

maintain charge neutrality. For any two edge-sharing Pd atoms with the same *z* coordinate and in the same bundle, the -Pd-Te-Pd- chains may be arranged such that these atoms are in the same or different coordination states; which of these is correct is indistinguishable from fits of the models to the room temperature x-ray PDF (R_w = 0.097 vs R_w = 0.098). Chemically, the second possibility is more plausible because the Te atoms which are axial ligands to these Pd atoms are also equitorial ligands for a third Pd. In order to displace the Te atoms without disrupting the equatorial plane of the Pd to which they are both bonded, one must displace up, and the other down. This has the effect of rotating Pd octahedra off of the *c*- axis.

Final parameters for the local structure at room temperature are given in Table 2.2. The magnitude of the displacement of I atoms in the *c*- direction is substantially larger than the displacement of Te atoms. The consequence of the vertical displacement of the iodine atoms is to introduce a non-zero torsional angle between the Te_2 and I_2 units of a $PdTe_2I_2$ square. Similar non-zero torsional angles are seen in a related, nominal Pd^{4+} compound, $PdTeI_2$.[21] Here, the non-zero torsional angle reflects the lower local symmetry present due to Pd^{2+}/Pd^{4+} charge ordering.

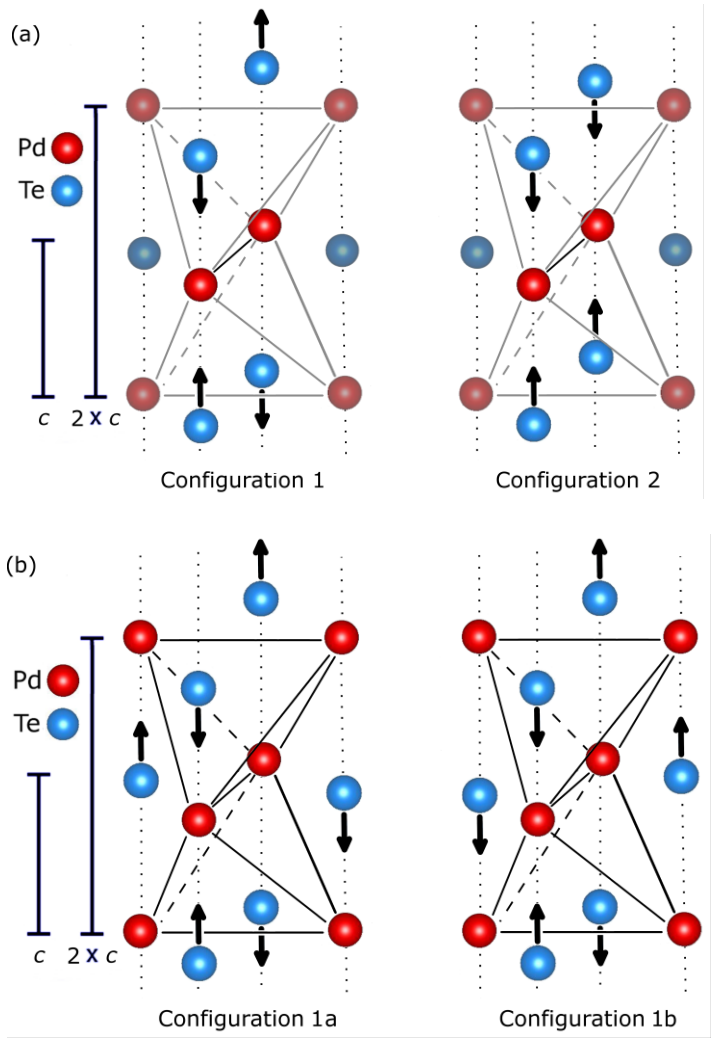


Fig. 2.10 (a) Within a bundle of -Pd-Te-Pd- chains (chains indicated by stippled lines), Pd atoms form Pd_4 tetrahedra. The displacement of Te atoms along the *c*- direction causes Pd atoms to approach square planar or octahedra coordination. For the pairs of chains with Pd on the same *ab* plane, two configurations are possible: one in which the square planar and octahedral sites are aligned (left), and one in which square planar sites are aligned with each other (right). The latter is more probable for geometric reasons. **(b)** Even with a unique selection of alignments of pairs of chains, the tetrahedral structure gives rise to two iso-energetic but symmetry-inequivalent conformations of the distorted -Pd-Te-Pd- chains.

PdTeI		T = 300 K		a = 7.80 Å;	c = 11.29 Å	
Atom	Site	x	y	z	occ.	U_{iso} (Å^2)
Pd1	4*a*	0	0.747	0	1	0.012
Pd2	4*a*	0	0.253	0	1	0.012
Te	8*d*	0	0.784	0.245	1	0.011
I	8*d*	0.242	0.5	0.987	1	0.010

Table 2.2 Parameters for a model of the local structure of PdTeI at room temperature obtained from a refinement to powder x-ray total scattering data to r_{max} = 7 Å, showing atomic positions, occupancies, and isotropic thermal parameters (U_{iso}). Space group $P4_322$ (95). The R_w for this refinement was 0.097.

Those features in the x-ray PDF which are not well fit by the average structure are also present in the neutron PDF data (Fig. 2.11). Several features of the neutron PDF are significantly broadened at higher temperature, to a greater extent than expected due to thermal motion alone. In particular, the peak at 5.8 Å, which corresponds to a Te-I distance, is completely smoothed at T = 300 K, while neighboring peaks show only slight broadening. The extra broadening originates from the quantum mechanical uncertainty in energy and time of a particle-atom interaction and suggests that the distortion of PdTeI structure is present above T_{CDW}. For the interaction of a thermal neutron with an atom the measured position of the atom will be a distribution over a timescale of ~10^{-12} s. The broadening of the 5.8 Å feature indicates that at 300 K the coordination state of Pd atoms is changing from pseudo-square planar to pseudo-octahedral with a timescale less than

$\sim 10^{-12}$ s. For higher energy synchrotron x-rays the interaction timescale is $\sim 10^{-15}$ s. Because the 5.8 Å feature is well defined in the x-ray PDF data we can determine that the timescale of these dynamics at 300 K is between $\sim 10^{-15}$ and $\sim 10^{-12}$ s.

It is non-trivial to extract precise correlation lengths of the charge density wave from the present data. This is due to the fact that the peak broadening in the SXRD data depends not only on the correlation length in different crystallographic directions but on the magnitude of the distortion giving rise to the CDW, preventing the use of Williamson-Hall and similar analysis techniques. However, an approximate average correlation length can be obtained from a plot of the residual R_w of fits of the average structure to the neutron PDF as a function of the r-range of the fit (Fig. 2.12). The quality-of-fit reaches a plateau as the model is fit to larger r_{max} ($r_{initial}$ = 1.5 Å for all plots). If we assume that the correlation length along a single chain is large, then the contribution of these *c*- aligned correlations to the quality-of-fit is expected to be relatively constant with increasing $r_{\max}$. The decrease in the quality-of-fit with decreasing r_{max} then implies a relatively short correlation length of 6-8 Å between chains.

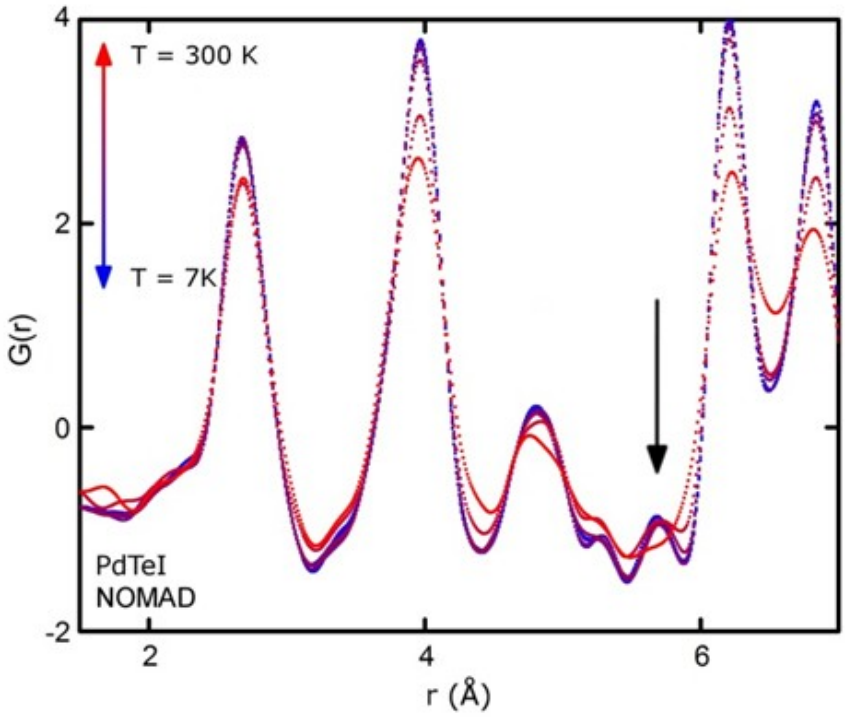


Fig. 2.11 The PDF of PdTeI as extracted from neutron total scattering data at a series of temperatures. The single-headed arrow indicates an atom-atom distance which is almost completely broadened at $T = 300$ K, in contrast to neighboring peaks which only show the slight broadening expected from thermal motion.

The short correlation length is not due to the presence small amounts of impurities. The channels in the structure formed by I atoms are sufficiently large to accommodate either K^+ or H^+ ions which would serve as an electron dopant. The terminations of coherent domains of the local structure of PdTeI consist of pairs of octahedra that are either both pseudo-square planar (Pd^{2+}) or both pseudo-octahedral (Pd^{4+}). Electron doping by K or H has the effect of increasing the number of such Pd^{2+}-Pd^{2+} pairs. EDX and PND refinements suggest a K^+/H^+ content of at most ~1 mol%. In the limit in which

all domain terminations arise from doping and none are thermally populated, the average domain size would be ~500 Å.

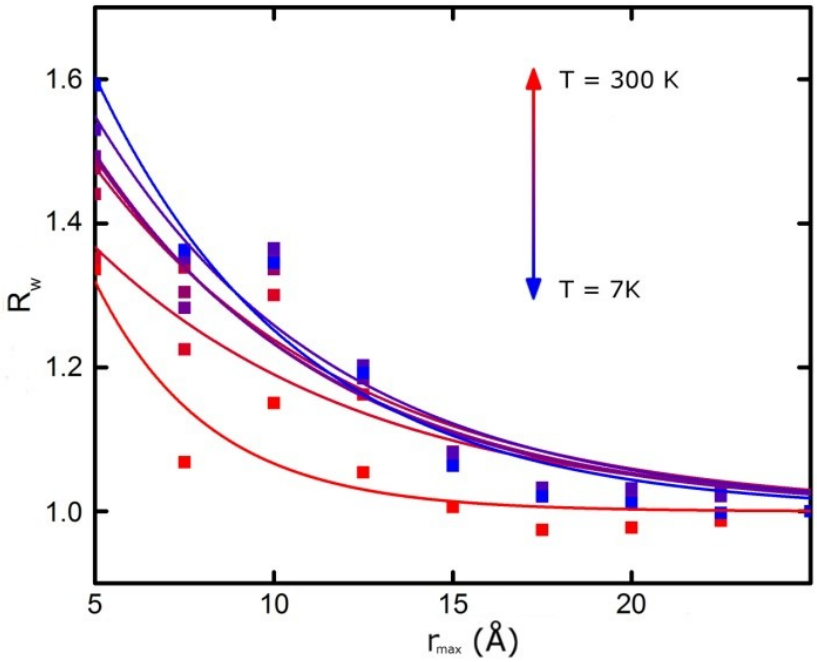


Fig. 2.12 The residual R_w from fits of the average structure to the neutron PDF as a function of r_{max} of the fit. R_w values for each temperature were scaled to the r_{max} = 25 Å values. All lines are guides to the eye. The sharp decay of R_w indicates that a local structure not described by the average structure has a correlation length of 6-8 Å at all temperatures.

A more probable reason for why short-range correlations between chains do not develop into long range order is the tetrahedral arrangement of Pd atoms within a bundle. Just like in frustrated tetrahedral spin systems, there are two inequivalent but isoenergetic configurations of the tetrahedra.[22] In this case, they correspond to different relative alignments of charge order between chains within a single bundle (Fig. 2.10b). This degeneracy leads to a frustration of charge order comparable to the frustration of spin in

pyrochlore antiferromagnets.[23] The result is small correlation lengths in the *a*- and *b*-directions (between chains). The frozen phase is then glass-like with no periodic order perpendicular to the chains.

Other charge-frustrated materials are known which are either charge-ordered, such as $LuFe_2O_4$,[24] or lack long-range charge order, such as $CaYNb_2O_7$.[25] Examples of CDW materials in which lower-symmetry domains of short coherence length give rise to a higher-symmetry average structures in Bragg diffraction experiments are well-known, including $BaFe_2Se_3$, $BaBiO_3$, KNi_2Se_2, KFe_2Se_3, KNi_2S_2, and $V_{1-x}Mo_xO_2$.[26,27,28,29,30] Metallic electrons are known to play a role in facilitating charge disorder via screening effects, such as in $Bi_2Ru_2O_7$,[31] and may be partly responsible for the lack of long-range order in PdTeI.

2.6 A Model of CDW Freezing

Taken together, the scattering data can be used to construct an electronic phase diagram for PdTeI (Fig. 2.13). Our data are consistent with the presence of dynamic charge fluctuations at room temperature that freeze out on cooling. The plateau in *hkl*-dependent peak broadening occurs around the same temperature as the sharp upturn in resistivity at T_{CDW}. We attribute this to freezing of the movement of chains relative to one another. Neighboring chains are arranged in one of two local configurations, without apparent long-range order. This arrangement disrupts lattice planes in the frozen phase and leads to the broadening of [*00l*] peaks in a manner similar to the effect of stacking faults in 2D materials. As the temperature increases above T_{CDW} the local order between chains becomes dynamic. As the temperature increases further, the charge ordering

within individual chains melts. However, analyses of the x-ray and neutron PDF indicate that at T = 300 K Pd atoms locally remain in distinct pseudo-square planar and pseudo-octahedral coordination environments and that these coordination environments retain short-range correlation, implying that charge fluctuations are still present and significant on the picosecond time scale.

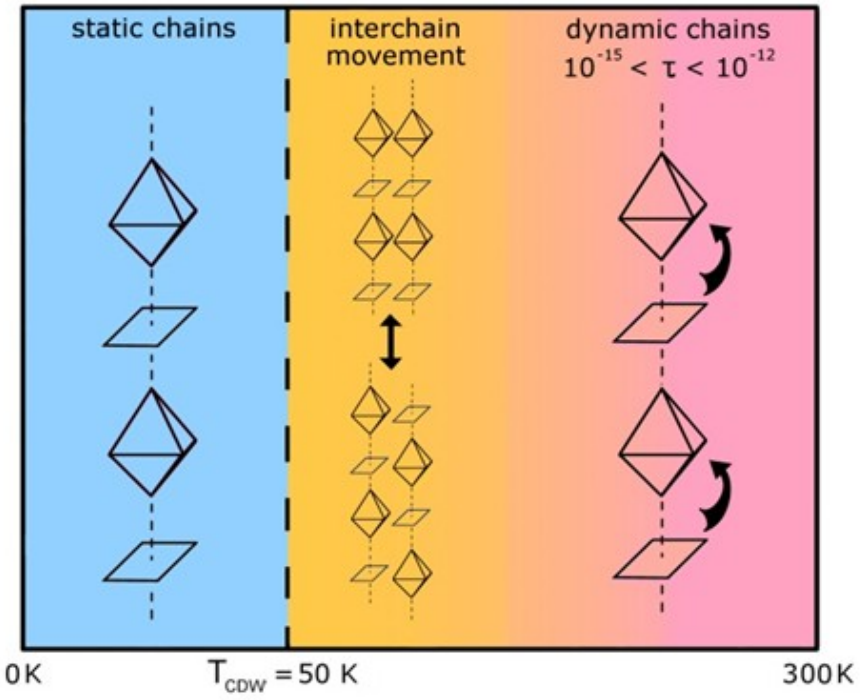


Fig. 2.13 Electronic phase diagram of PdTeI. Below T_{CDW}, individual Pd atoms are statically charge disproportionated along a chain as Pd^{2+}/pseudo-square planar and Pd^{4+}/pseudo-octahedral, but without long range order perpendicular to the chain direction. Above T_{CDW}, chains are able to move relative to one another, followed by dynamical motion within each chain on further warming.

2.7 Conclusions

These results indicate that palladium in PdTeI spontaneously disproportionates towards Pd^{2+} and Pd^{4+} rather than existing in the highly unusual Pd^{3+} oxidation state. This disproportionation leads to the formation of a 1D CDW and a local distortion causing alternating Pd atoms in the chains to develop pseudo-square planar and pseudo-octahedral coordination. The distortion persists in a dynamic form above the CDW melting temperature. The charge-ordering between adjacent -Pd-Te-Pd- chains within a bundle has multiple lowest-energy configurations. This 'frustration' prevents the formation of a long range charge-ordered state. Similar effects may be at play in compounds such as the hotly debated rare earth nickelates and other reported compounds with elements in unusual oxidation states. It will be interesting to probe how such charge frustration impacts carrier dynamics and other electrical properties in PdTeI and related materials.

2.8 References

(1) Thiele, G.; Köhler-Degner, M.; Wittmann, K.; Zoubek, G. *Agnew. Chem. Int. Ed. Engl.*, **1978**, *17*, 852-853.

(2) Mirica, L.; Khusnutdinova, J. *Coord. Chem. Rev.* **2013**, *7*, 299-314.

(3) Kim, S. -J.; Lemaux, S.; Demazeau, G.; Kim, J. -Y.; Choy, J. -H. *J. Am. Chem. Soc.* **2001**, *123*, 10413.

(4) Tressaud, A.; Khairoun, S.; Dance, J. M.; Hagenmuller, P. *Z. Anorg. Allg. Chem.*, **1984**, *517*, 43- 58.

(5) P. Day, *Low-dimensional Cooperative Phenomena*, ed. H. J. Heller Plenum, New York, **1975**.

(6) Goodenough, J. B. *Rep. Prog. Phys.* **2004**, *67*, 1915-1993.

(7) Piamonteze, C.; Tolentino, H. C. N.; Ramos, A. Y.; Massa, N. E.; Alonso, J. A.; Martinez-Lope, M. J.; Casais, M. T. *Phys. Rev. B* **2005**, *71*, 012104.

(8) Seo, D. K.; Whangbo, M. H.; Neininger, K.; Thiele, G. *J. of Sol. State Chem.,***1998**, *137*, 206-210.

(9) Larson, A. C.; Von Dreele, R. B. *Los Alamos National Laboratory Report LAUR* **2000**, 86-748.

(10) Toby, B. H. *J. Appl. Crystallogr*. **2001**, *34*, 210-213.

(11) Qiu, X., Thompson, J. W.; Billinge, S. J. L. *J. Appl. Crystallogr.* **2004**, *37*, 678.

(12) Farrow, C. L.; Juhás, P.; Liu, J.; Bryndin, D.; Bozin, E. S.; Bloch, J.; Proffen, Th.; Billinge, S. J. L. *J. Phys: Condens. Mat.* **2007**, *19*, 335219.

(13) F. J. Di Salvo, R. G. Maines, J. V. Waszczak, and R. E. Schwall, *Sol. Stat. Comm.* **1974**, *14*, 497-501.

(14) Harper, J. M. E.; Geballe, T. H.; Di Salvo, F. J. *Phys. Lett. A* **1975**, *54*, 27-28.

(15) Lue, C. S.; Kuo, Y. -K.; Hsu, F. H.; Li, H. H.; Yang, H. D.; Fodor, P. S.; Wenger, L. E. *Phys. Rev.B* **2002**, *66*, 033101.

(16) Smontara, A.; Biljakovic, K.; Mazuer, J.; Moncea, P.; Levy, F. *J. Phys.: Condens. Matter*. **1992**, *4*, 3273-3281.

(17) Ashcroft, N.; Mermin, N. *Solid State Physics*, Brooks/Cole, Belmont CA, 1976.

(18) Mott, N. F. *Rev. Mod. Phys.* **1949**, *32*, 677-682.

(19) Wilson, K. G. *Rev. Mod. Phys.*, **1975**, *47*, 773-840.

(20) Daniels, P. *J. Appl. Cryst.*, **1998**, *31*, 559-569.

(21) Brendel, K. PhD thesis, Albert Ludwigs Universität, **2001**.

(22) Pauling, L. *J. Am. Chem. Soc.* **1935**, *57*, 2680-2684.

(23) Harris, M. J.; and M. P. Zinkin, *Mod. Phys. Lett. B* **1996**, *10*, 417-438.

(24) Ikeda, N.; Ohsumi, H.; Ishii, K.; Inami, T.; Kakurai, K.; Murakami, Y.; Yoshii, K.; Mori, S.; Horibe, Y.; Kito, H. *Nature* **2005**, *436*, 1136-1138.

(25) McQueen, T. M.; West, D. V.; Muegge, B.; Huang, W.; Noble, K.; Zandbergen, H. W.; Cava, R. J. *J. Phys. Cond. Matt.* **2008**, *20*, 235210.

(26) Caron, J. M.; Neilson, J. R.; Miller, D. C.; Arpino, K.; Llobet, A.; McQueen, T. M. *Phys. Rev. B* **2012***, 85***,** 180405.

(27) Neilson, J. R.; Llobet, A.; Steir, A. V.; Wen, L.; Tao, J.; Tesanovic, Z. B.; Armitage, N. P.; McQueen, T. M. *Phys. Rev. B,* **2012**, *86*, 054512.

(28) Rice, J.; Wang, Y. *Physica C* **1989,** *157***,** 192-197.

(29) Neilson, J. R.; Llobet, A.; Wen, J. -J.; Suchomel, M. R.; McQueen, T. M. *Phys. Rev. B*, **2013**, *87*, 045124.

(30) Holman, K. L.; McQueen, T. M.; Williams, A. J.; Klimczuk, T.; Stephens, P. W.; Zandbergen, H. W.; Xu, Q.; Ronning, F.; Cava, R. J. *Phys. Rev. B.* **2009**, *79*, 245114.

(31) Shoemaker, D. P.; Sheshadri, R.; Tachibana, M.; Hector, A. L. *Phys. Rev. B.***2011**, *84*, 064117.

Chapter 3: Instrumentation for Photovoltage and Photocurrent Spectroscopy with sub-0.1 mW/cm^2 Irradiation from 350 nm $\leq \lambda \leq$ 1700 nm and from 1.8 K $\leq T \leq$ 300 K

This work was co-written with the following authors and is published under the following citation:

Patrick Cottingham, Amanda Lemire, Penny Lemire, and Tyrel M. McQueen

in submission

3.1 Introduction

Photoconductivity is a property of materials that is extremely important to the performance of photodiodes, solar cells, photocatalysts, and other technologies. Photoconductivity measurements can be used to study optical bandgaps[1], carrier lifetimes[2], and other important electronic properties of materials. Measuring photoconductivity as a function of temperature and wavelength is a powerful method for investigating complex band structures[3] and the energies and densities of defects in a material.[4] A related property, the photomagnetoresistance, has been used to study the structure of conduction bands in insulators[5] and spin-dependent recombination.[6]

Photoconductivity was discovered serendipitously when a selenium resistor was exposed to ambient light during an electronic measurement.[7] In order to resolve features in photoconductivity spectra, modern photoconductivity experiments often use a monochromatic light source such as a collection of LEDs,[8] monochromated lamp,[9] or laser.[10] Unlike LEDs or traditional lasers, monochromated lamp sources can produce light over a broad and continuous wavelength range, albeit with relatively less power.

The use of low power illumination in photoconductivity measurements is advantageous for several reasons. Light energy absorbed by the device under test (sample) and its environment is converted into heat, which limits the lowest achievable measurement temperature. Large irradiances can also complicate the accurate measurement of bulk sample temperature during a measurement, for example, by creating temperature gradients within a sample. Further, low light power also helps avoid contaminating thermoelectric effects that stem from temperature gradients. Reduced temperatures can also improve the sensitivity of measurements that involve low-energy transitions by reducing the number of carriers that undergo those transitions by thermal excitation.

Additionally, frequency-chopped low-intensity light sources can be combined with larger continuous irradiation light sources in order to probe different regions of a non-linear photoresponse. For example, the optical generation of carriers varies the quasi-Fermi level of a semiconductor, which may change how defects function as either traps or recombination centers.[11] The continuous light source may be used to set the quasi-Fermi level, while the response to a lower-intensity frequency-chopped source is measured in order learn about the behavior of defects for a given Fermi level.

In this article we present the technical details and theory of operation of instrumentation for measuring the photoconductivity of materials and photocurrents generated by devices under monochromatic illumination with wavelength 350 nm $\leq \lambda \leq$ 1700 nm and total incident light power $P < 30$ μW at temperatures 1.8 K $\leq T \leq$ 300 K.

3.2 Instrument Details

3.2.1 Optical and Measurement Components

The instrument for measuring photoconductivity is shown in Fig. 3.1. A 100 W quartz-tungsten halogen (QTH) lamp (a) is used to produce white light that enters collimating optics (b). The collimated light is modulated by an optical chopper (c) and passes through a filter wheel (d) that eliminates higher harmonics after the beam is monochromated. The beam enters through adjustable slits (e) into a dual-grating monochromator (f) that selects a narrow spectral band from the incoming white light. The use of two gratings increases the range of output wavelengths. The partial overlap in the wavelength range of each grating allows for measurements to be made at a single wavelength using each grating, in order to eliminate the possibility of artifacts due to differences in angular dispersion and beam width between the two gratings. The beam exits the monochromator through an additional set of slits which may be adjusted to reduce both the size and the spectral width of the exiting beam. The exiting beam is collimated again and impinges on a beam splitter, which divides the beam into perpendicular fractions with ~10% and ~90% of the original intensity, respectively.

The more intense fraction is focused by a lens (h) onto a spherical sapphire lens (i) which further focuses the beam onto the aperture of an uncoated silica optical fiber. This fiber is butt-coupled to a gold-coated, low-temperature- and vacuum-resistant optical fiber contained in a probe (k), which may be inserted into the dewar of a Physical Properties Measurement System (PPMS) (not shown). The dewar is cooled by the evaporation of liquid helium to reach temperatures as low as T = 1.8 K. The PPMS also

contains a super-conducting magnetic coil which may be used to apply a magnetic field of up to $B = 9$ T.

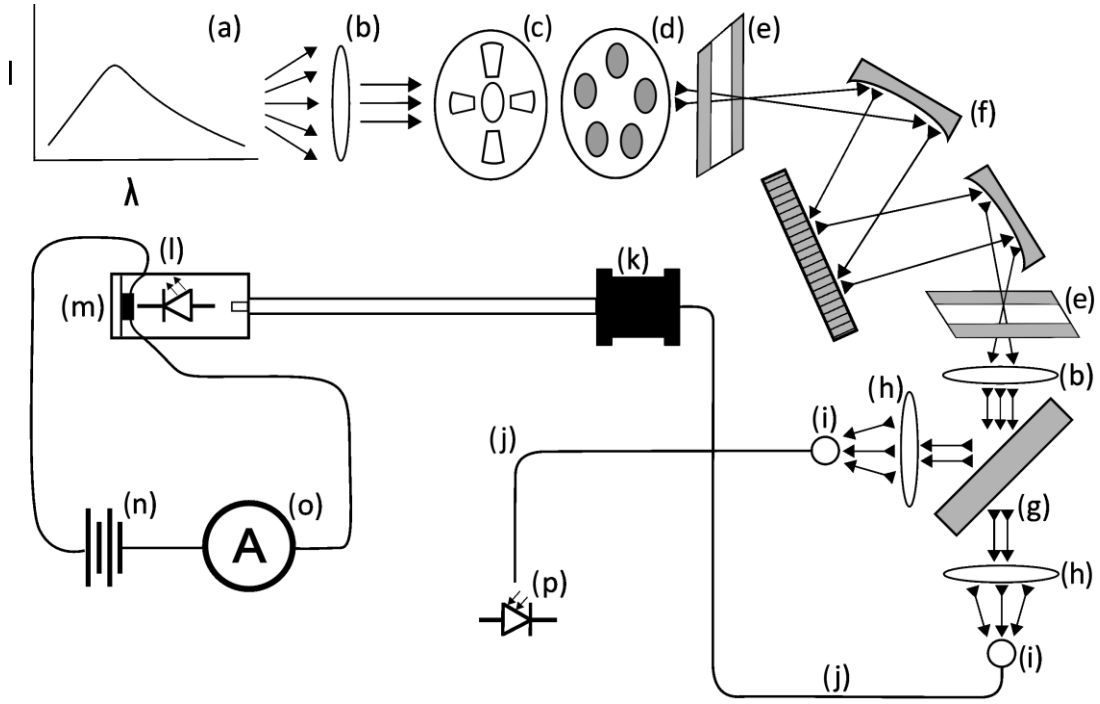


Fig. 3.1 Schematic of the instrument. Individual components are described in the main text.

The sample stage at the bottom of the probe may be fitted with an LED (l) which is driven by a power source outside of the PPMS (not shown) via electrical contacts at the bottom of the probe. Light from optical fiber and/or the LED illuminate a sample (m) which is connected to external measurement electronics. I_0 delivered to the sample from the fiber varies with wavelength and QTH lamp setting, and is less than 15 μW at all wavelengths when chopping and less than 30 μW when not chopping. LEDs may be used to provide illumination at a single wavelength with I_0 limited by the specifications of the LED chosen. The light from lamp/monochromator and the LED may be

modulated independently of one another or constant (in the case of the LED, output is modulated by modulating the electrical power to the LED).

The sample may be connected to the desired measurement electronics and current or voltage supplies via the contacts on the probe. Depicted in Fig. 3.1 are a voltage source (n) and an ammeter (o), used to collect two-probe photoconductivity measurements.

The less intense fraction exiting the beam splitter is focused into an optical fiber via an optical path identical to the one leading to the probe. Light exiting this fiber illuminates a photodiode (p) which is connected to a power meter (not shown). The power measured by this sensor is recorded simultaneously with measurements of the sample. The recorded values are multiplied by functions which account for the wavelength-dependent efficiency of the beam splitter and attenuation by the fiber in the probe in order to calculate I_0 illuminating the sample. Not shown is a small piece of graphite which is taped to the bottom of the probe (k) in order to getter helium in the chamber at low temperatures – we found that liquid helium can condense on samples and distort the illumination.

The power supply, power meter, monochromator, lock-in amplifier, and power supply for LEDs are collectively controlled by custom software, which allows for automation and the coordination of events in the course of a series of measurements. This code may be operated independently or it may interface with the Multivu software published by Quantum Design in order to additionally control the temperature and magnetic field. The custom software was written in C++ and the original components of the code are given in appendix A.

Components (a)-(h), (p), and the power meter were purchased from the Newport Corporation. Spherical sapphire lenses were purchased from Edmund Optics, Inc. All optical fibers were purchased from Fiberguide Industries, Inc. Many combinations of instruments may be used for electronic measurements of samples mounted in the probe; the instruments used to collect data in the following sections were a Stanford Research Instruments SRS830 lock-in amplifier, a Marlin P. Jones and Associates, Inc. 14604PS power supply, and the resistivity option of a PPMS manufactured by Quantum Design, Inc. The multi-function probe (MFP) which was modified to produce the probe in (k) was also manufactured by Quantum Design, Inc.

3.2.2 Sample Stage

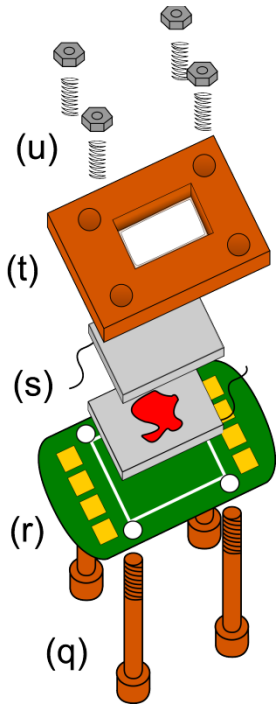


Fig. 3.2 Schematic of a device for mounting thin film and powder samples

For macroscopic samples, electrical contacts can be made by attaching leads to the sample with conducting paste and attaching the leads to electrical contacts on the

sample stage with solder. Hydrocarbon grease may be used to immobilize the sample and increase thermal contact with the sample stage.

Thin films and powders may be mounted on the sample stage using the apparatus shown in Fig. 3.2 Flanged and threaded posts made of oxygen-free high-conductivity (OFHC) Cu (q) are inserted through holes drilled into the sample stage (r). Thin film samples which are deposited onto a conducting substrate are placed against a transparent conducting glass upper electrode with conducting side facing the sample (s). The substrate and the upper electrode are connected to the electrical contacts on the sample stage in order to form a measurement circuit.

For a powdered sample, the sample is placed on top of a bottom electrode such as foil or conducting glass and the upper electrode is carefully placed on top. For both thin films and powders, the upper and lower electrode are compressed against an OFHC Cu plate with a window (t) using screws and nuts (u). OFHC Cu was chosen for (q) and (t) because it is non-magnetic and highly thermally conducting.

3.2.3 Calculation of Correction Functions

In order to calculate I_0 incident on the sample using the values recorded from the photodiode (p) it is crucial to determine a correction function $F(\lambda) = I_0 / I_{diode}$ where I_{diode} is the light power striking the photodiode when it is illuminated with the less intense fraction of the light exiting the beam splitter. This function should account for both the wavelength-dependent efficiency of the beam-splitter (g) and for the different amounts of attenuation in the optical paths leading to the sample and the photodiode, respectively. The function may be determined simply by measuring I_0 for all relevant λ by positioning

the photodiode in the bottom of the probe (k) in a darkened chamber, and subsequently measuring I_{diode} for the same series of λ.

Ideally, at least the factors related to the beam-splitter should be measured whenever the lamp is replaced or repositioned because small differences in the position of the filament can affect these values. Given the frequent need to perform these measurements and the slightly cumbersome process of measuring I_0, it may advantageous to perform a one-time measurement of I_0 closely followed by a measurement of I_0', the light power exiting the fiber from the more intense fraction exiting the beam splitter. It is important to closely space these measurements in order to minimize the contribution of lamp aging on the ratio of measured I_0 to measured I_0'. Separate, closely spaced measurements of I_0' and I_{diode} can be made whenever the lamp is changed or adjusted.

The two ratios I_0 / I_0' and I_0' / I_{diode} can then be used to find $F(\lambda)$ as follows:

$$F(\lambda) = {I_0}/{I_0'} \cdot {I_0'}/{I_{diode}} . \qquad \textbf{(3.1)}$$

Two typical plots of I_0 / I_0' and I_0' / I_{diode} are shown in Fig. 3.3 (a) and (b) respectively.

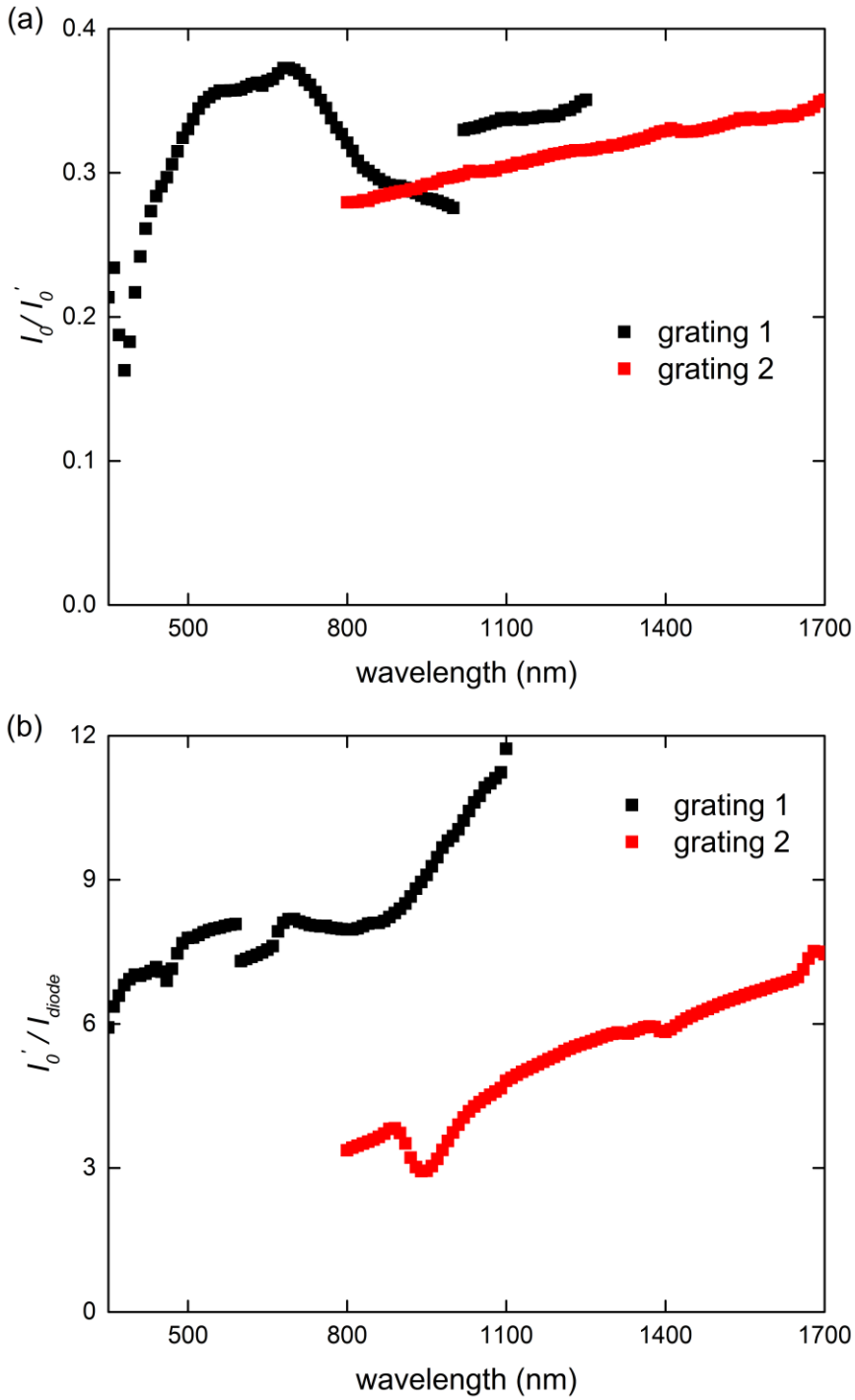


Fig 3.3 (a) The ratio I_0/I_0' as a function of wavelength for grating 1 and grating 2. The large discontinuity at λ = 1000 nm corresponds with a change in filter. **(b)** A representative plot of I_0'/I_{diode} as a function of wavelength for grating 1 and grating 2. This spectral shape changes slightly whenever a new lamp is installed.

3.2.4 Spectral Width

The resolution of any measurement will be limited by the bandpass (the spectral width), *B*, of the monochromated light source. For large *B* relative to the resolution of a monochromator, *B* is given by

$$B = {W \cdot a \cdot \cos\theta_D}/{f \cdot m}. \quad \textbf{(3.2)}$$

where *W* is the width of the exit slits, *a* is the ruling line width of the grating, θ_D is the diffraction angle, *f* is the focal length of the monochromator, and *m* is the order of diffraction.

3.2.5 Correction for LED performance

If an LED mounted on the sample stage is used, then it is important to know the performance of the LED at each measurement temperature. Relative changes in performance can be determined by directing the LED output upwards through the probe-mounted fiber and measuring with the photodiode. Absolute output of the LED at room temperature can be determined by mounting the photodiode on the sample stage with the photodiode masked to reflect the area of the sample to be measured. The amount of intensity reaching the sample from the LED at a given temperature can be calculated by adjusting the room-temperature intensity by the relative variation in LED performance with temperature.

3.2.6 Temperature Measurement

Temperature may be measured simultaneously with photoconductivity measurements using thermocouples mounted in the base of the PPMS dewar and in the probe. In order to confirm that the temperature of these thermocouples accurately

represents the temperature of the sample when illuminated by the lamp, the resistivity of vanadium was measured near its superconducting transition at $T_C = 5.30$ K[12] (Fig. 3.4a). The resistivity of the sample decreased to zero when the temperature of the thermocouples is between 5.1 K < *T* < 5.3 K, indicating that the temperature of the probe is within 0.2 K of the sample temperature at 5.3 K. Illuminating the vanadium sample with $\lambda = 650$ nm or $\lambda = 1050$ nm light did not decrease the probe temperature at which the superconducting transition occurs, suggesting that heating due to illumination is negligible.

We also performed the same measurement on a crystal of SmB_6, a blue-black insulator (Fig. 3.4b). The resistivity vs. temperature curves for SmB_6 in the dark and under illumination with $\lambda = 650$ nm or $\lambda = 1050$ nm light overlap. This indicates that heating[13] due to illumination is also negligible for strongly absorbing materials with reduced thermal conduction between the irradiated area and the probe.

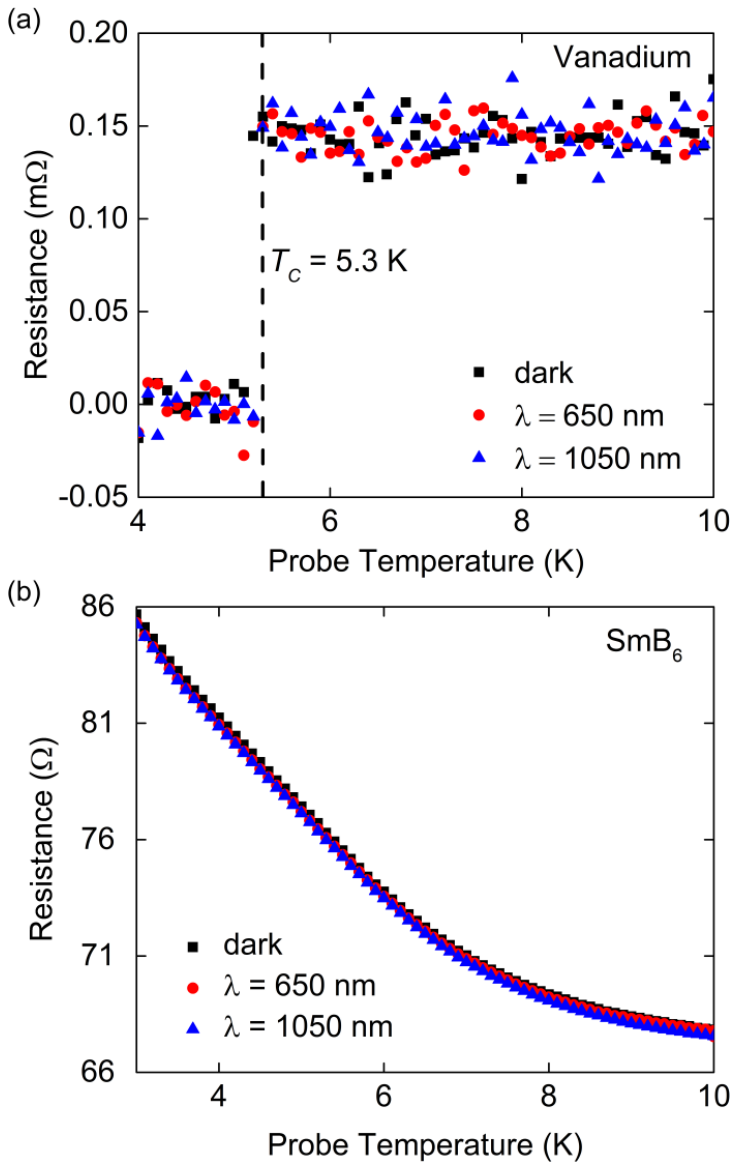


Fig. 3.4 The resistivity of **(a)** V, and **(b)** SmB_6, mounted in the photoconductivity probe, in the dark and under illumination.

3.3 Measurement Configurations

3.3.1 Applied voltage, measured current

The determination of $\Delta\sigma$ can be accomplished by applying a constant current and measuring a change in voltage, ΔV, or applying a constant voltage and measuring changes

in current, Δi. In order to isolate the response of the sample/device from the responses of other circuit elements, it is important to understand the circuits used in these measurements. For reader convenience, circuits are analyzed in terms of a changes in resistance ΔR which can then be easily converted into conductance, ΔG.

ΔR is given by

$$\Delta R = R_{\text{light}} - R_{\text{dark}} \quad (\mathbf{3.3})$$

where R_{light} is the resistance the sample when illuminated and R_{dark} is the resistance of the sample when it is in the dark. R_{light} and R_{dark} are the inverses of the conductances G_{light} and G_{dark}, respectively. ΔR can be rewritten as

$$\Delta R = \frac{G_{\text{dark}} - G_{\text{light}}}{G_{light} G_{\text{dark}}}, \quad (\mathbf{3.4})$$

which re-arranges to

$$\Delta R = -\Delta G_S R_{\text{light}} R_{dark,} \quad (\mathbf{3.5})$$

where ΔG is the difference of G_{light} and G_{dark} (the photoconductance). ΔG is related to the photoconductivity $\Delta\sigma$ by

$$\Delta\sigma = \Delta G \cdot {}^{l}/_{A} \quad (\mathbf{3.6})$$

where A and l are the area and length of the sample.

For samples with a resistance which is large relative to the resistance of contacts made to the sample, it convenient to measure photoconductivity using a two-probe, constant-voltage configuration (Fig. 3.5).

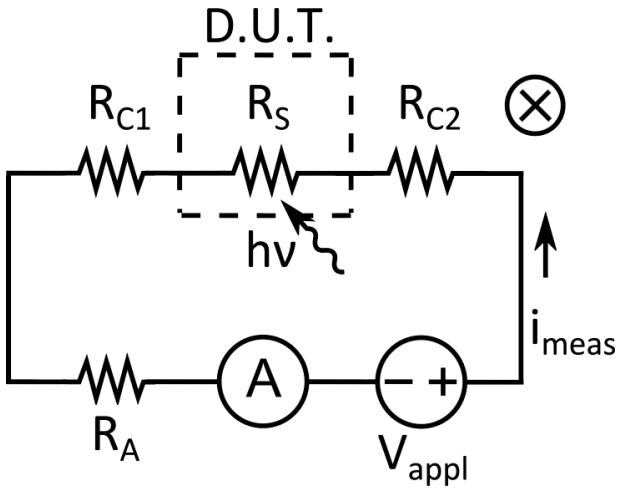


Fig. 3.5 A circuit for measuring the photoconductivity of a sample using a two-probe, constant-voltage configuration.

In this configuration, two leads are attached with a conducting paste to opposite sides of a sample. The resistances of the two contacts and the sample when it is not being illuminated with the modulated source are designated R_{C1}, R_{C2}, and $R_{S,dark}$, respectively. The sample is connected in series with an ammeter (the lock-in amplifier in current mode) with internal resistance R_A and a constant voltage source supplying V_{appl}. In Figs 3.5 - 3.7 D.U.T. (Device under Test) identifies the sample, and the symbol in upper right hand corner indicates a magnetic field going into the page. The field may be applied in either direction along the indicated axis. For the sake of convenience, R_{C1}, R_{C2}, R_A and $R_{S,dark}$ are combined here into R_{dark}:

$$R_{dark} = R_{S,dark} + R_{C1} + R_{C2} + R_{A.} \qquad (3.7)$$

When the modulated light source is off, the current though the circuit is i_{dark}, which is given by Ohm's law:

$$V_{appl} = i_{dark} R_{dark}. \qquad (3.8)$$

When the sample is irradiated, the resistance changes by ΔR_S, causing the current to change by Δi . Because V_{appl} remains constant, this leads to the expression

$$i_{\mathrm{dark}} R_{\mathrm{dark}} = (i_{\mathrm{dark}} + \Delta i)(R_{\mathrm{dark}} + \Delta R_S) \qquad (\mathbf{3.9})$$

which can be re-arranged to give the desired quantity ΔR_S:

$$\Delta R_S = -\frac{\Delta i R_{\mathrm{dark}}}{(\Delta i + i_{\mathrm{dark}})}. \qquad (\mathbf{3.10})$$

ΔR_S determined in this way can be inserted into Eqn. 3.5 in order to find ΔG_S with the result:

$$\Delta G_S = {}^{\Delta i}/_{V_{\mathrm{appl}}}. \qquad (\mathbf{3.11})$$

3.3.2 Constant Current, Measured Voltage

For samples which are relatively conducting compared to the contacts and the internal impedance of the amplifier, it is useful to measure the photoconductivity of the sample using a four-probe, constant-current configuration. In this configuration, four contacts designated C1 ... C4 with resistances R_{C1} ... R_{C4} are made to the sample. Ideally, the sample should be in a bar shape and the contacts should be arrange in a line. A constant current, i_{appl}, is supplied through the outermost contacts (C1 and C4). The change in voltage upon illumination, ΔV, across the innermost contacts (C2 and C3) is measured with the lock-in amplifier in voltage mode. This configuration has the advantage of minimizing contributions due to contacts because i_2, the current flowing through the voltmeter, is much smaller than i_1, the current flowing through the sample.

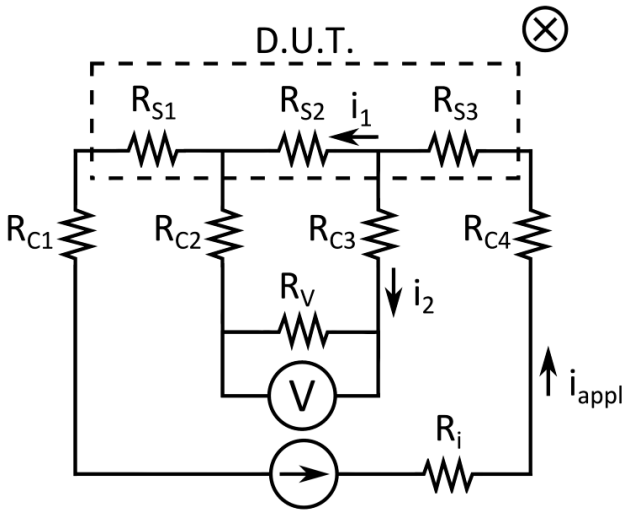


Fig. 3.6 A circuit for measuring the photoconductivity of a sample using a four-probe, constant-current configuration.

The currents in Fig. 3.6 are related by the equations

$$i_{appl} = i_1 + i_2 \qquad (\mathbf{3.12})$$

and

$$i_1 R_{S2} = i_2(R_{C2} + R_V + R_{C3}). \qquad (\mathbf{3.13})$$

If i_1 obtained from Eqn. 3.13 is substituted into Eqn. 3.12, an expression for R_{S2} is obtained:

$$R_{S2} = \frac{i_2(R_{C2} + R_{C3} + R_V)}{i_{appl} - i_2}. \qquad (\mathbf{3.14})$$

When the sample is irradiated, i_2 changes from its dark value of $i_{2,\,\mathrm{dark}}$ to a new value of $i_{2,\mathrm{dark}} + \Delta i_2$. Inserting these values into Eqn. 3.14 and subtracting the values of R_{S2} before and after illumination yields the equation

$$\Delta R_{S2} = \frac{(\Delta i_2 + i_2)(R_{C2} + R_{C3} + R_V)}{i_{appl} - i_2 - \Delta i_2} - \frac{i_2(R_{C2} + R_{C3} + R_V)}{i_{appl} - i_2} \qquad (\mathbf{3.15})$$

which describes ΔR_{S2}, the change in R_{S2} when the sample is illuminated. Using the assumptions that $i_{appl} \gg i_2$ and $R_V \gg R_{C2}$, R_{C3} to simplify Eqn. 3.15, and substituting ΔV for $\Delta i_2 \cdot R$V, we obtain

$$\Delta R_{S2} = \frac{\Delta V}{i_{appl}} \qquad (\mathbf{3.16})$$

which gives ΔR_{S2} in terms of known and measured variables. In order to calculate the photoconductivity ΔG_{S2}, it is necessary to measure the additional value V, the voltage drop across C2 and C3 when the sample is not being illuminated. This can be measured simply by connecting temporarily connecting a conventional voltmeter in place of the lock-in amplifier. Combining Eqn. 3.16 with Eqn. 3.5 gives the desired expression for ΔG_{S2}:

$$\Delta G_{S2} = {-\Delta V \cdot i_{appl}}\Big/{(V^2 + V\Delta V)}. \qquad (\mathbf{3.17})$$

3.3.3 Measurement of PV Devices

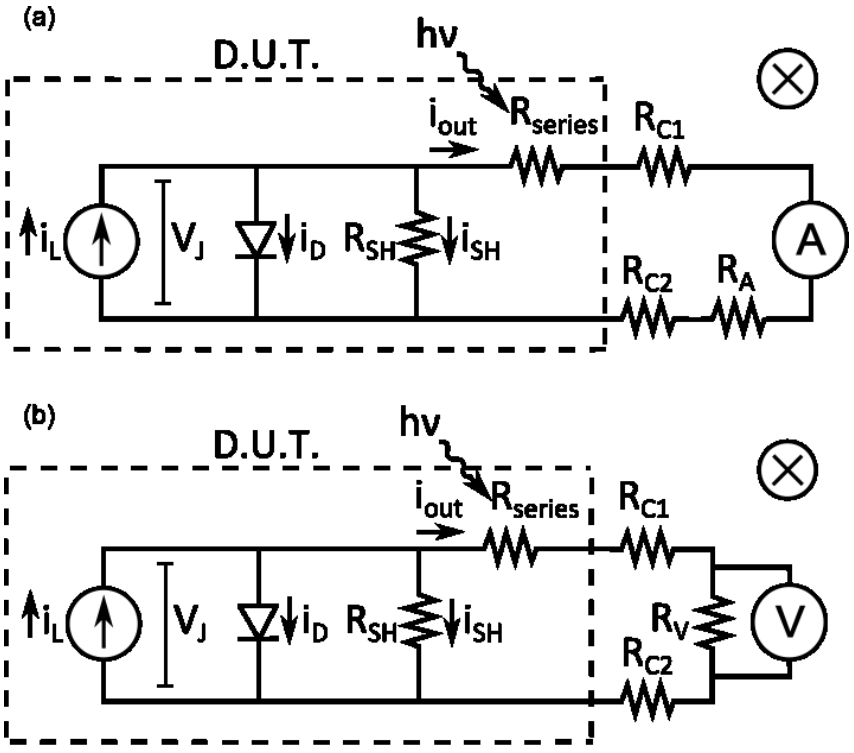


Fig. 3.7 Circuits for measuring **(a)** the short-circuit current and **(b)** the open-circuit voltage of phovoltaic devices.

Photovoltaic (PV) devices may be measured without the application of any applied current or voltage. When the solar cell is connected to the lock-in amplifier in current mode (Fig. 3.7a), the measured current will be given by a slight modification of the characteristic equation of solar cells

$$i_{out} = i_L - i_0\left(e^{\frac{qi_{out}(R_{series}+R_{C1}+R_{C2}+R_A)}{nkT}} - 1\right) - \frac{i_{out}(R_{series}+R_{C1}+R_{C2}+R_A)}{R_{SH}} \quad (\mathbf{3.18})$$

which has been changed to include resistance of the lock-in amplifier, V_A, and the contacts to the device, R_{C1} and R_{C2}. In Eqn. 3.18 i_{out} is the current delivered by the device,

i_0 is the reverse saturation current, q is the fundamental charge, R_{series} is the series resistance to the output current due to the device itself, k is the Boltzmann constant, T is temperature, and R_{SH} is the shunt resistance, a value related to current losses within the solar cell. The second term of the equation describes the diode current, which is indicated by i_D in Fig. 3.7.

i_L is the photogenerated current, which describes the rate of generation of free charge carriers in the device, similar to $f(\lambda)$ for materials. When R_{series}, R_{C1}, R_{C2}, and R_A are small, i_{out} is referred to as i_{SC} which is approximately equal to i_L. When the sample is in the dark, all terms in Eqn. 3.18 are equal to zero, so Δi measured when the sample is irradiated with modulated light is equal to i_{SC}.

V_{OC} is the voltage developed by a PV device when its output terminals are disconnected. V_{OC} does not relate simply to any intrinsic property of a material, but is determined by the choice of materials and the architecture of a device. V_{OC} is approximately equal to the voltage measured when the device is connected to a high-impedance voltmeter as shown in Fig. 3.7b.

Any of the configurations shown in Figs. 3.5-3.7 can be used with an applied magnetic field. The photo-magnetoresistance, $\Delta\rho/\rho$, is the relative change in resistance that occurs in an irradiated sample when a magnetic fields is switched on. If the two-probe, constant voltage configuration is used with a DC ammeter subsituted in place of the lock-in amplifier, then $\Delta\rho/\rho$ will be given by

$$\Delta\rho/\rho = (i_H - i_O)/i_O \qquad (\mathbf{3.19})$$

where i_H is the current measured with an applied field and i_O is the current measured with no applied field. If the four-probe, constant-current configuration is used with a DC

voltmeter in place of the lock-in amplifier, then Eqn. 3.19 will hold with the analogous voltages V_H, and V_O substituted for i_H and i_O, respectively.

3.4 Example Data

3.4.1 Photoconductivity of Instrinsic Silicon

In order to confirm that the instrument works as intended, the photoconductivity of two previously well studied materials, intrinsic Si and β-In_2S_3 were measured. Fig. 3.8a shows the photoconductivity of a single crystal of intrinsic silicon, measured in two-probe, constant-voltage mode using the monochromated light source. V_{appl} for this measurement was 1 V, and Ohmic contacts to the sample were made by pressing heated indium pads against the sample.

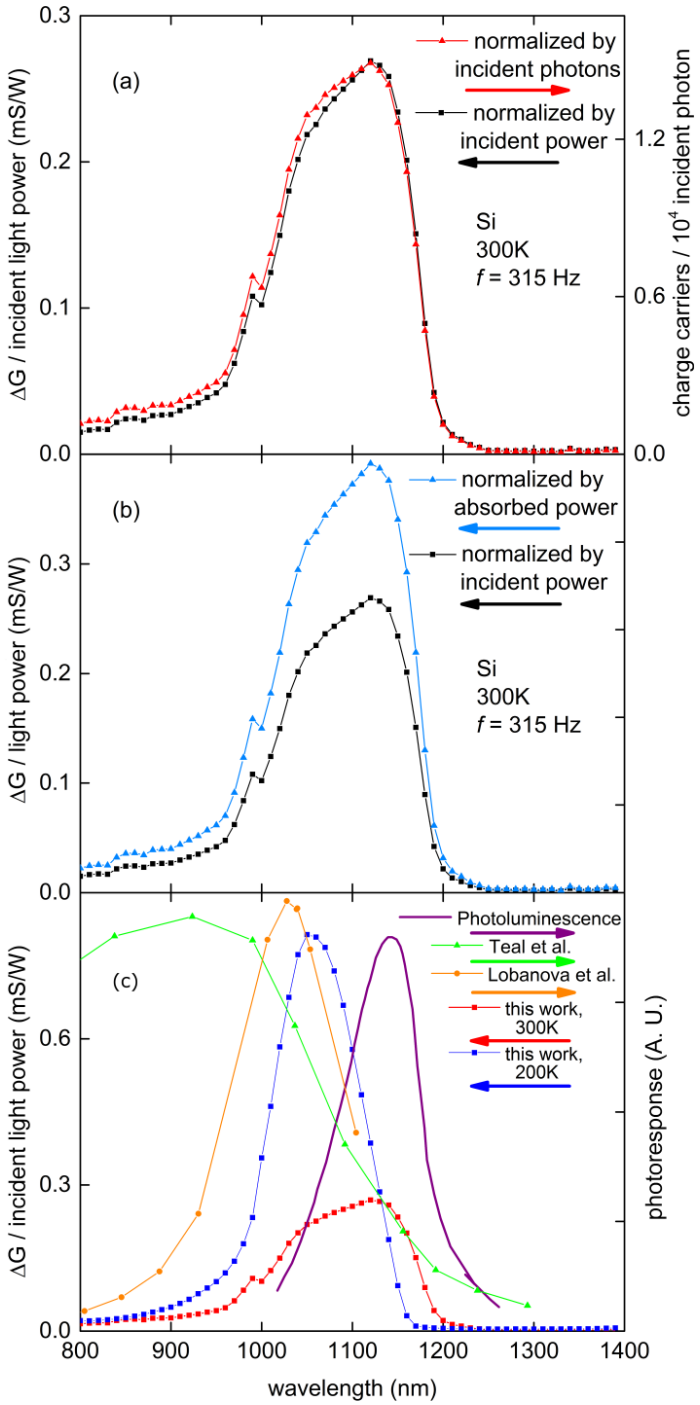


Fig. 3.8 (a) The photoconductivity of intrinsic silicon normalized by incident light power and number of incident photons **(b)** photoconductivity of silicon normalized by incident light power and absorbed light power **(c)** comparison of four measurements of the photoconductivity of silicon and the photoluminescence of silicon

The two curves illustrate two ways that the photoconductivity can be normalized: by incident light power and by the number of incident photons. The difference in the two curves is related to the decreasing number of photons per unit of light power as the wavelength decreases. In general, the appropriate choice of normalization scheme for a given measurement will depend on the physical properties being studied. Using literature values for the reflectance of intrinsic silicon,[14] the data may also be corrected for the number of absorbed photons or the absorbed light power (Fig. 3.8b).

These data are in general agreement with previously published measurements of the photoconductivity of silicon[15,16,17] (Fig. 3.8c). All of the data feature a peak-like response and band edge between 1100 nm < λ < 1200 nm. The photoconductivity of silicon is highly affected by doping, and differences between the data collected for this work and older data are likely the result of impurities present in samples produced via older refinement techniques. Absorption and emission processes near the indirect band edge are phonon-mediated, and thus highly sensitive to temperature. Our photoconductivity spectrum at T = 300 K appears to contain two features, which are centered at the same wavelengths as the single feature of the photoconductivity at T = 200 K and the room temperature photoluminescence. This indicates that the T = 300 K spectrum contains two phonon-mediated transitions, in agreement with previous measurements and with expectations based on the band structure.

Because the photoconductivity of silicon is known to vary non-linearly with light power,[16] it is especially important to confirm that apparent changes with wavelength are not due to the change in the power of the light source with changing wavelength. The

un-normalized photoconductance of a silicon sample under illumination with λ = 1040 nm at a series of light powers is shown in Fig. 3.9. The variation in power was achieved by varying the power supplied to the lamp and by using each of the two gratings to select the desired wavelength. The two sets of closely spaced power values correspond to measurements taken using the different gratings. The data show that the un-normalized photoconductivity of intrinsic silicon at λ = 1040 nm varies nearly linearly with power over the range of powers achievable with the instrument at that wavelength (non-linearity at other wavelengths cannot be excluded).

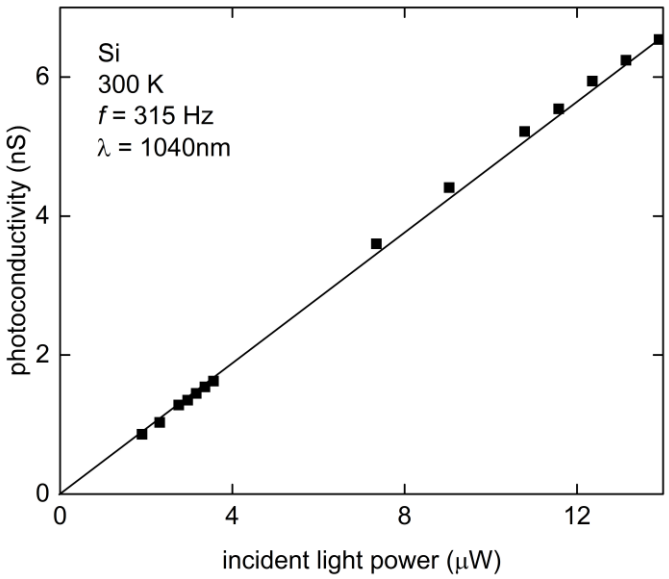


Fig. 3.9 Photoconductivity of silicon vs. incident light power.

The photoconductivity of intrinsic silicon as a function of temperature is shown in Fig. 3.10. As the temperature of the sample decreases, the spectral response narrows and the band edge shifts to shorter wavelengths. This is consistent with silicon having an indirect bandgap. As the temperature of the sample is decreased, higher-energy phonon modes are depopulated. Because interaction with a phonon is necessary for an electron to

undergo an indirect transition, the spectral response narrows due to the decreased number of populated phonon modes.

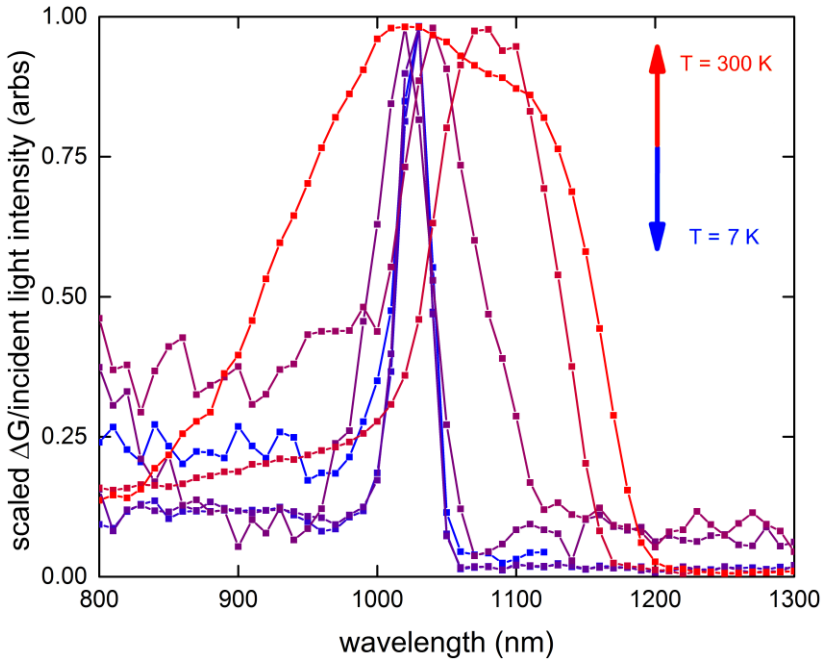


Fig. 3.10 The photoconductivity of intrinsic Si at a series of temperatures. All spectra are scaled such that the maximum signal of each spectrum is the same.

3.4.2 Photoconductivity of β-In_2S_3

Fig. 3.11a shows the photoconductivity of a sample of the direct-bandgap semiconductor β-In_2S_3. These data were collected using the two-probe, constant-voltage configuration and the monochromated light source. The un-normalized photoconductance spectrum of β-In_2S_3 measured using our instrument is similar to data published by Ho et al.[3] which show a peak-like response centered near λ = 600 nm. When our data is normalized by the incident light power, the normalized photoconductance displays the spectral shape typically associated with a direct-bandgap semiconductor. This shape consists of a sharp edge near λ = 700 nm corresponding to the

~2.1 eV bandgap of In_2S_3, a local upturn near the photoconductivity edge, and relatively a flat response at shorter wavelengths.

The overall scale of this spectrum increases when the chopping speed is decreased (Fig. 3.11b) but the spectral shape remains the same. This dependence of the scale of the photoconductance on chopping frequency is indicative of some free-carrier relaxation process that has a characteristic timescale similar to the period of the modulation of the light source. Fitting models of relaxation to the changes in photoconductance with frequency can provide insight into these timescales.[18]

3.5 Conclusions

A novel instrument for measuring the photoconductivity and photomagnetoresistance of materials and devices over a large, continuous range of wavelengths and temperatures was developed. In order to correctly translate measured values into physically meaningful parameters of the sample alone it is crucial to consider the response of the measurement circuit, the light power incident on the sample, and non-linear and frequency-dependent effects. Methodologies to account for these effects have been provided.

The capabilities of this instrument lend themselves to the application of photoconductivity to many current areas of research, including phenomena beyond the simple physics discussed above. This instrument has recently been used to measure the temperature-dependence of multiple distinct features in the photocurrent spectrum of $CH_3NH_3PbI_3$:TiO_2 -based solar cells.[19] Differences in the temperature-dependence of

these features strongly suggest the existence of multiple charge separation pathways in these devices.

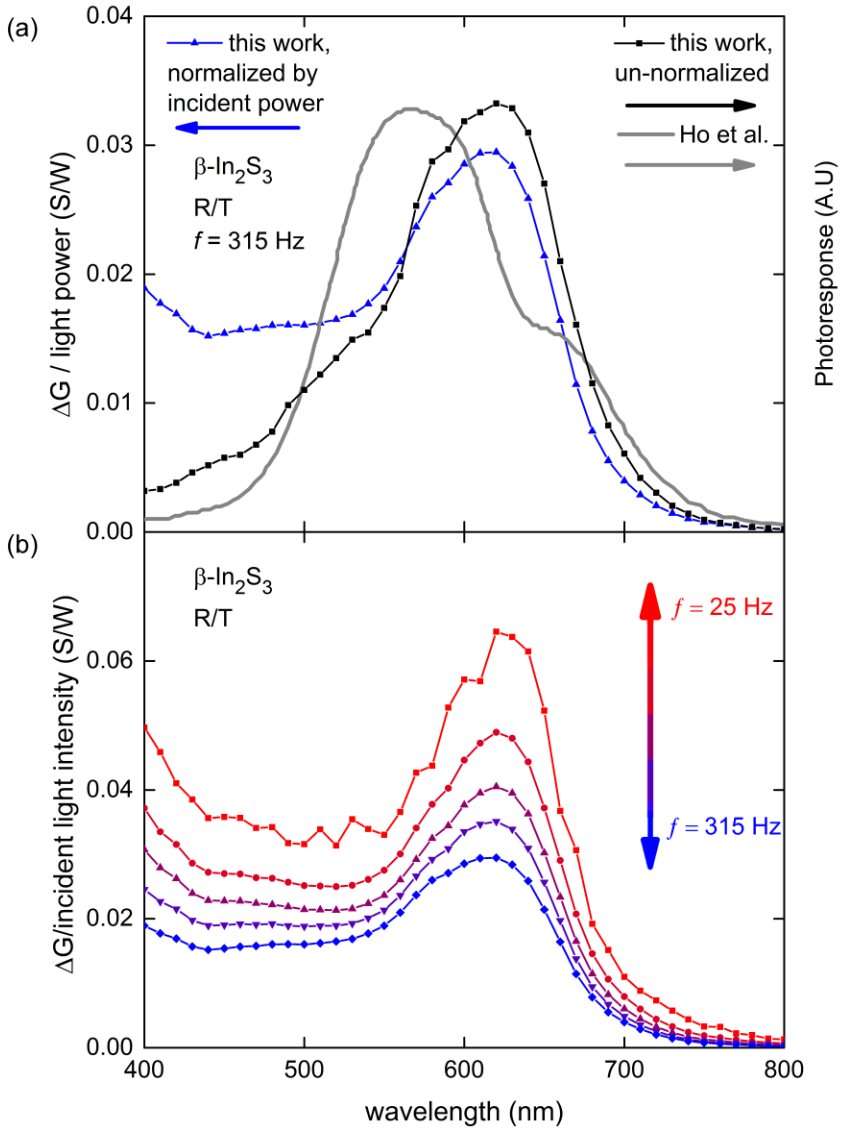


Fig. 3.11 (a) The raw photoresponse and incident power normalized photoconductance of In_2S_3 measured in this work and the photoresponse measured by Ho et al. **(b)** The incident power normalized photoconductance of In_2S_3 measured at several frequencies

Studies of ZnO nanowires have shown they possess a strongly temperature-dependent persistent photoconductivity (PPC).[20, 21] This PPC has been attributed to the

thermal trapping of carriers at low temperatures.[21] Our instrument is well suited for investigating the depth and distribution of the relevant trap states by populating the states with bright LED illumination at low temperatures and subsequently re-exciting trapped carriers with sub-bandgap illumination from the monochromated source.

Illumination with visible or infrared light can induce complicated changes in the behavior of charge density waves (CDWs) which can be studied with photoconductivity measurements.[22,23,24,25] These include reduction of the threshold field for CDW sliding and light-induced switching behavior. Our instrument is well-suited for these measurements, especially in materials with a low CDW transition temperature, T_{CDW}.

3.6 References

(1) Piper, W. W. *Phys. Rev*. **1953**, *92*, 23-27.

(2) Mayburg, S. *Rev. Sci. Instrum*. **1953**, *26*, 616-617.

(3) Ho, C. H.; Yang, Y. P.; Chan, C. H.; Huang, Y. S.; Li, C. H. *J. Appl. Phys*. **2010**, *108*, 043518.

(4) Qasrawi, A. F.; Gasanly, A. M. *Semicond. Sci. Technol*. **2005**, *20*, 446-452.

(5) Yasunaga, H. *J. Phys*. *Soc. Jpn*. **1968,** *24*, 1308-1313.

(6) Lindström, M.; Kuivalainen, P.; Heleskivi, J.; Galazka, R. R. *Physica B+C* **1983** *117-118*, 479481.

(7) Smith, W. *Nature* **1873**, *7*, 303.

(8) Montenegro, R.; Inocente-Junior, N. R.; Frejlich, J. *Rev. Sci. Instrum*. **2006**, *77*, 043905.

(9) Tyndall, E. P. T. *Phys. Rev*. **1923**, *21*, 162.

(10) Hotchkiss, D. R. *J. Appl. Phys*. **1964**, *35*, 2455-2457.

(11) Bube, R. *Photoconductivity of Solids*. John Wiley & Sons, Inc.: New York, 1963.

(12) Däumer, W.; Kettschau, A.; Khan, H. R.; Lüders, K.; Raub, Ch. J.; Riesemeier, H.; Roth, G.; *Phys. Stat. Sol*. **1982**, *112*, 67-71.

(13) Phelan, W. A.; Koohpayeh, S. M.; Cottingham, P.; Freeland, J. W.; Leiner, J. C.; Broholm, C. L.; McQueen, T. M. *Phys. Rev. X* **2014**, *4*, 031012.

(14) Green, M. A. *Sol. Energy Mater. Sol. Cells* **2008**, *92*, 1305-1310.

(15) Teal, J. K.; Fisher, J. R.; Treptow, A. W. *J. Appl. Phys*. **1946**, *17*, 879-886.

(16) Petrusevich, V. A.; Lobanova, T. N. *Sov. Phys. Solid State* **1962,** *3*, 2575-2577.

(17) Haynes, J. R.; Westphal, W. C. *Phys. Rev*. **1956**, *101*, 1676.

(18) Lee, C. H.; Yu, G.; Heeger, A. J. *Phys. Rev. B* **1993**, *47*, 15543.

(19) Cottingham, P.; Wallace, D. C.; Hu, K.; Meyer, G.; McQueen, T. M. *Chem. Commun*. **2015**, *51*, 7309-7312.

(20) Prades, J. D.; Hernandez-Ramirex, F.; Jimenez-Diaz, R.; Manzanares, M.; Andreu, T.; Cirera, A.; Romano-Rodriguex, A.; Morante, J. R. *Nanotechnology* **2008**, *19*, 465501.

(21) Liao, Z.-M.; Lu, Y.; Xu, J.; Zhang, J.-M.; Yu, D.-P. *Appl. Phys. A* **2009**, *95*, 363-366.

(22) Latyshev, Y. I.; Minakova, V. E.; Savitskaya, Y. S.; Frolov, V. V. *Physica B+C* **1986**, *143*, 155-177.

(23) Zaitsev-Zotov, S. V.; Minakova, V. E. *JETP Lett*. **2004**, *79*, 550-554.

(24) Zaitsev-Zotov, S. V.; Minakova, V. E. *Phys. Rev. Lett*. **2006**, *97*, 266404.

(25) Zaitsev-Zotov, S. V.; Nasretdinova, V. F.; Minakova, V. E. *Physica B* **2015**, *460*, 174-179.

3.7 Appendix A: Instrument Control Software

3.7.1 Header Files

"config. h"

```
#include "stdafx.h"
#include <stdio.h>

#define MC_LOG_FILE "MCControlv9.log"
#define MC_DATA_FILE "MCData.dat"
#define MC_SEQUENCE_FILE "MCSequence.seq"
#define GPIB_READ_TERM_CHAR        0xA

int FindWhiteSpace (LPTSTR toTerminate);
bool InitializeConfigPointer(LPTSTR toinit, LPTSTR inputstr);
bool StripConfigString (LPTSTR configArg);
bool ReadConfigFile();

struct configuration
{
       char* gpib_write_default_sleep; //ms
       char* gpib_comm_mutex_timeout; //ms
       char* gpib_board;
       char* use_ppms;
       char* general_wait_time;
       char* ppms_gpib_address;

// Use 300W Power Supply? False indicates no, True indicates yes
       char* use_power_supply;

//Power Supply Configuration Options
       char* ps_initial_power;
       char* ps_port;
       char* ps_stepsize;
       char* ps_default_write_sleep;


// Do we use the Cornerstone 260 monochromator or not? False for no, True for yes
       char* use_cornerstone260;

// Cornerstone 260 configuration options
       char* cs_gpib; //GPIB address for CS monochromator
       char* cs_stepsize;       //nm
       char* cs_grating1_start; // nm
       char* cs_grating1_max;   // nm
       char* cs_grating2_start;      // nm
       char* cs_grating2_max;      // nm
       char* cs_write_sleep_grating_change;  // ms
```

```
        char* cs_write_sleep_filter_change;    // ms
        char* cs_write_sleep_gowave;           // ms
        char* cs_filter_1_max;    // nm
        char* cs_filter_2_max;    // nm
        char* cs_filter_3_max;    // nm
        char* cs_filter_4_max;    // nm

// Do we use the resistivity bridge read options or not? Uncomment to use.
        char* use_ppms_resistivity_bridge;

// PPMS resistivity bridge configuration options
        char* rb_num_readings;
        char* rb_wait_between_readings;

// Reading which bridges?
        char* rb_read_bridge_1;
        char* rb_read_bridge_2;
        char* rb_read_bridge_3;

// how many bridges being read;
        char* rb_num_bridge_read;

// Do we use the SRS830
        char* use_SRS830;
        char* SRS830_gpib;
        char* SRS830_chan;
        char* SRS830_auto_gain;
        char* SRS830_write_sleep;

// Do we use the NOVA power meter
        char* use_NOVA;
        char* NOVA_meas_sleep;
        char* NOVA_port;
        char* NOVA_default_write_sleep;


// Do we use the K2400
        char* use_k2400;
        char* k2400_meas_sleep;
        char* k2400_port;
};

"CornerStone260.cpp"
// Cornerstone 260 Advises from PPMS
#define CS_MIN_ADVISE                   1
#define CS_ADVISE_RESET_GRATING_1       1
#define CS_ADVISE_RESET_GRATING_2       2
#define CS_ADVISE_INCRWL                3
#define CS_ADVISE_CLOSESHUTTER          4
#define CS_ADVISE_OPENSHUTTER           5
#define CS_MAX_ADVISE                   7


////////////////////////////////////////////////////////////////////////////
// Cornerstone 260 global variables
////////////////////////////////////////////////////////////////////////////

// Cornerstone 260
```

```
extern bool CS_initialized;
extern int CS_addr;
extern int CS_dev;
//first wavelength in a sweep
extern int CS_grating1_startwaveln;
extern int CS_grating2_startwaveln;
////output wavelength increment size in nm
extern int CS_stepsize;
//current wavelength
extern int CS_curwaveln;
//filter currently in place
extern int CS_curfilter;
// current grating. Using grating 1 for wl < GRATING_CHANGE_WL, grating 2
otherwise
extern int CS_curgrating;
// current shutter 0 = closed; 1 = open
extern int CS_curshutter;

////////////////////////////////////////////////////////////////////////////

bool CS_init();
bool CS_do(int advise);
bool CS_shutter(bool openit);
bool gratingswitch(int newgrating);
bool gotowavelength(int targetwln);

"k2400.h"

// Advises for k2400

#define k2400_MIN_ADVISE          40
#define k2400_MEASURE             41
#define k2400_MAX_ADVISE          49

////////////////////////////////////////////////////////////////////////////
// 300W Power Supply global variables
////////////////////////////////////////////////////////////////////////////
extern bool k2400_initialized;
//points to serial port
extern HANDLE k2400Serial;
//last value read
extern double k2400_last_value;


////////////////////////////////////////////////////////////////////////////


bool k2400_measure();
bool k2400_do(int advise);
bool k2400_init();

"MCControl9.h"

#include "stdafx.h"

// GPIB & PPMS Base Functions
#include ".\gpib\ni488.h"
#include ".\ppms\ppmsuser.h"
```

```
// GPIB & PPMS User Functions
#include "ppmsutil.h"

// Useful function not found on windows
#define round(f_val) (( f_val < 0.0f ) ? (int)(f_val - 0.5) : (int)(f_val + 0.5))

// Configuration options
#include "config.h"

// Newport Optics Cornerstone 260 Monochromator (GPIB version)
#include "CornerStone260.h"

// Stanford Research Instruments 830 lock-in amplifier
#include "SRS_830.h"

// Newport Optics Power Meter (USB version)
#include "PowerMeter.h"

// Power Supply for Light Source (via RS-232)  (Base functions are defined in
windows.h)
#include "PowerSupply.h"

// keithley 2400 source-meter (via RS-232)
#include "k2400.h"

//NOVA power meter (via RS-232)
#include "NOVA.h"

// load and run sequence file (for use when operating independent of the PPMS)
#include "sequence.h"



#define ADVISORY_WRITE_MEASUREMENT 20
#define GENERAL_WAIT_ADVISE 26

// GO_BACK advises are used to implement loops.  Last two digits gives number of
steps back to go in the sequence.
// 2nd-4th digits are number of repetitions
#define GO_BACK_MIN_ADVISE 100001
#define GO_BACK_MAX_ADVISE 199999

//ADVISES 9990-9999 are reserved


////////////////////////////////////////////////////////////////////////////
//Common global variables
////////////////////////////////////////////////////////////////////////////

// Log File
extern FILE *logFile;

// Data File
extern FILE *dataFile;

// Session identifier
```

```
extern int MC_uid;

// configuration
extern configuration config;

////////////////////////////////////////////////////////////////////////////

"NOVA.h"

#define NOVA_MIN_ADVISE           50
#define NOVA_MEASURE              51
#define NOVA_MAX_ADVISE           59

////////////////////////////////////////////////////////////////////////////
// 300W Power Supply global variables
////////////////////////////////////////////////////////////////////////////
extern bool NOVA_initialized;
//points to serial port
extern HANDLE NovaSerial;
//last value read
extern double NOVA_last_value;


////////////////////////////////////////////////////////////////////////////


bool NOVA_measure();
bool NOVA_do(int advise);
bool NOVA_init();

"PowerSupply.h"

// Power Supply Advises from PPMS

#define PS_MIN_ADVISE           21
#define PS_ADVISE_RESETPOW      21
#define PS_ADVISE_INCRPOW       22
#define PS_ADVISE_LAMP_OFF      23
#define PS_ADVISE_LAMP_ON       24
#define PS_MAX_ADVISE           24

////////////////////////////////////////////////////////////////////////////
// 300W Power Supply global variables
////////////////////////////////////////////////////////////////////////////
extern bool PS_initialized;
//points to serial port
extern HANDLE hSerial;
//first power in a sweep
extern int PS_initial_power;
//power increment size in Watts
extern int PS_stepsize;
//current power
extern int PS_curpower;
// is the lamp on or off?  FALSE for off, TRUE for on
extern bool PS_lamp_state;
////////////////////////////////////////////////////////////////////////////

bool PS_read_lamp_state();
```

```
bool PS_change_lamp_state(bool desiredstate);
bool PS_change_power(int power);
bool PS_do(int advise);
bool PS_init();
```

"ppmsuutil.h"

```
// All "set advise" commands in PPMS Multivu result in a WM_USER event
// the wParam then indicates the actual advise number.
#define PPMS_ADVISORY  WM_USER


// PPMS GPIB send and receive
long PPMSSendAndReceive(char *cmd, char *response, int responseSize);

// PPMS-safe GPIB read and write
bool ibwrite(int ud, char *cmd, int sleep);
bool ibread(int ud, char *s, long cnt);
int ibdevice(int pad, int sad, int tmo, int eot, int eos);
void WriteGPIBBoardConfig();

// PPMS record measurement
void PPMSWriteDataHeader();
bool PPMSWriteMeasurement();
```

"sequence.h"

```
#include "stdafx.h"
#include <stdio.h>

#define MC_SEQUENCE_FILE "MCSequence.seq"


extern long *seqlist;
extern long *seqlist_original;

bool ReadSeqFile();
```

"SRS_830.h"

```
// Standford Research Systems 830 Advises from PPMS

////////////////////////////////////////////////////////////////////////////
// Standford Research Systems 830 global variables
////////////////////////////////////////////////////////////////////////////

// Standford Research Systems 830
extern bool SRS830_initialized;
extern int SRS830_addr;
extern int SRS830_dev;
extern float SRS830_last_chan1;
extern float SRS830_last_phase;
////////////////////////////////////////////////////////////////////////////

bool init_SRS830();
bool SRS830_auto_gain();
```

```
bool SRS830_reset();
bool SRS830_meas();
```

"stdafx.h"

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__A9DB83DB_A9FD_11D0_BFD1_444553540000__INCLUDED_)
#define AFX_STDAFX_H__A9DB83DB_A9FD_11D0_BFD1_444553540000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define WIN32_LEAN_AND_MEAN       // Exclude rarely-used stuff from Windows
headers

#pragma warning(disable: 4996)

#include <windows.h>
#include <atlbase.h>

#include <stdlib.h>
#include <stdio.h>
#include <time.h>

#endif // !defined(AFX_STDAFX_H__A9DB83DB_A9FD_11D0_BFD1_444553540000__INCLUDED_)
```

"targetver.h"

```
#pragma once

// Including SDKDDKVer.h defines the highest available Windows platform.

// If you wish to build your application for a previous Windows platform, include
WinSDKVer.h and
// set the _WIN32_WINNT macro to the platform you wish to support before including
SDKDDKVer.h.

#include <SDKDDKVer.h>
```

3.7.2 Source Files

"config.cpp"

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "MCControlv9.h"

#define CONFIG_MEM_BLOCK 4096

int FindWhiteSpace (char* toTerminate, int maxLength){
       //precondition:receives pointer to string with one or more white spaces
```

```
	//postcondition:returns position of first white space
	int k = 0;
	char *c = (char *)calloc(1, 32);
	if (toTerminate == NULL){
		fprintf(logFile, "FindWhiteSpace: curpos == Null, returning false
		\n");
		return -1;
	}
	*c = toTerminate[k];
	while (!(strncmp(c, " ", maxLength) == 0) && !(strncmp(c, "\n", maxLength)
		== 0)){
		k++;
		*c = toTerminate[k];
	}
	return k;
}


bool InitializeConfigPointer(char* toinit, char* inputstr, int maxLen){
	//precondition: recieves pointer to string containing the configuration
file, pointing at the first character of a char string to be loaded into the
memory location pointed to by toinit
	//postcondition: the memory location pointed to by toinit contains the char
string pointed to by inputstr, with the first whitespace char in the string now a
NULL character
	if (inputstr == NULL){
	 fprintf(logFile, "InitializeConfigPointer: inputstr is NULL, returning
		FALSE \n");
		return false;
	}
	if (toinit == NULL){
	fprintf(logFile, "InitializeConfigPointer: toinit is NULL, returning FALSE
		\n");
		return false;
	}
	if (maxLen < 1){
	fprintf(logFile, "InitializeConfigPointer: Invalid maxLen=%i, returning
		FALSE \n", maxLen);
		return false;
	}
	int firstspace = FindWhiteSpace(inputstr, maxLen);
	int k = 0;
	while  (k < firstspace && k < maxLen-1) {
		toinit[k] = inputstr[k];
		k++;
	}
	toinit[k] = NULL;
	return TRUE;
}

bool StripConfigString (char* configArg){
	//precondition: configArg contains the characters of config.ini
	//postcondition: memory is allocated for the LPTSTRs of struct config.  The
	memory pointed to by those LPTSTRs contains the initialization values,
	NULL-terminated
	if (configArg == NULL || strlen(configArg) < 1){
		fprintf(logFile, "StripConfigString: configArg == NULL \n");
		return false;
```

```
}
fprintf(logFile, "StripConfig_string: configArg is not NULL\n");

char *curArg = (char *)malloc(sizeof(configArg));
curArg = (strstr(configArg, "gpib_write_default_sleep=") + 25);
config.gpib_write_default_sleep = (char *)calloc(1, 32);
fprintf(logFile, "StripConfig_string: allocated memory to
       gpib_write_default_sleep\n");
if (!InitializeConfigPointer(config.gpib_write_default_sleep, curArg, 32))
fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned false
       when initializing gpib_write_default_sleep\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       gpib_write_default_sleep %s\n", config.gpib_write_default_sleep);
curArg = (strstr(configArg, "gpib_comm_mutex_timeout=") + 24);
config.gpib_comm_mutex_timeout = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.gpib_comm_mutex_timeout, curArg, 32))
fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned false
       when initializing gpib_comm_mutex_timeou t\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
gpib_comm_mutex_timeout %s \n", config.gpib_comm_mutex_timeout);
curArg = (strstr(configArg, "gpib_board=") + 11);
config.gpib_board = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.gpib_board, curArg, 32))
fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned false
       when initializing gpib_board\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated gpib_board %s \n",
       config.gpib_board);
curArg = (strstr(configArg, "use_ppms=") + 9);
config.use_ppms = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.use_ppms, curArg, 32)) fprintf(logFile,
       "StripConfigStr: InitializeConfigPointer returned false when
       initializing  use_ppms\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated use_ppms %s \n",
config.use_ppms);
curArg = (strstr(configArg, "general_wait_time=") + 18);
config.general_wait_time = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.general_wait_time, curArg, 32))
fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned false
       when initializing general_wait_time\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       general_wait_time %s \n", config.general_wait_time);
curArg = (strstr(configArg, "ppms_gpib_address=") + 18);
config.ppms_gpib_address = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.ppms_gpib_address, curArg, 32))
fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned false
       when initializing ppms_gpib_address\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       ppms_gpib_address %s \n", config.ppms_gpib_address);
curArg = (strstr(configArg, "use_power_supply=") + 17);
config.use_power_supply = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.use_power_supply, curArg, 32))
fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned false
       when initializing use_power_supply\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       use_power_supply %s \n", config.use_power_supply);
curArg = (strstr(configArg, "ps_initial_power=") + 17);
config.ps_initial_power = (char *)calloc(1, 32);
```

```
if (!InitializeConfigPointer(config.ps_initial_power, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing ps_initial_power\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       ps_initial_power %s \n", config.ps_initial_power);
curArg = (strstr(configArg, "ps_port=") + 8);
config.ps_port = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.ps_port, curArg, 32)) fprintf(logFile,
       "StripConfigStr: InitializeConfigPointer returned false when
       initializing ps_port\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated ps_port %s \n",
config.ps_port);
curArg = (strstr(configArg, "ps_stepsize=") + 12);
config.ps_stepsize = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.ps_stepsize, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing ps_stepsize\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated ps_stepsize %s
       \n", config.ps_stepsize);
curArg = (strstr(configArg, "ps_default_write_sleep=") + 23);
config.ps_default_write_sleep = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.ps_default_write_sleep, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing ps_default_write_sleep\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       ps_default_write_sleep %s \n", config.ps_default_write_sleep);
curArg = (strstr(configArg, "use_cornerstone260=") + 19);
config.use_cornerstone260 = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.use_cornerstone260, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing use_cornerstone260=\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       use_cornerstone260 %s \n", config.use_cornerstone260);
curArg = (strstr(configArg, "cs_gpib=") + 8);
config.cs_gpib = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.cs_gpib, curArg, 32)) fprintf(logFile,
       "StripConfigStr: InitializeConfigPointer returned false when
       initializing cs_gpib=\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated cs_gpib %s \n",
       config.cs_gpib);
curArg = (strstr(configArg, "cs_stepsize=") + 12);
config.cs_stepsize = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.cs_stepsize, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing cs_stepsize=\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated cs_stepsize %s
       \n", config.cs_stepsize);
curArg = (strstr(configArg, "cs_grating1_start=") + 18);
config.cs_grating1_start = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.cs_grating1_start, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing cs_grating1_start\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       cs_grating1_end_range %s \n", config.cs_grating1_start);
curArg = (strstr(configArg, "cs_grating1_max=") + 16);
config.cs_grating1_max = (char *)calloc(1, 32);
```

```
if (!InitializeConfigPointer(config.cs_grating1_max, curArg, 32))
fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned false
       when initializing cs_grating1_max=\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       cs_grating2_reset_wl %s \n", config.cs_grating1_max);
curArg = (strstr(configArg, "cs_grating2_start=") + 18);
config.cs_grating2_start = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.cs_grating2_start, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing cs_grating2_start=\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       cs_grating2_start %s \n", config.cs_grating2_start);
curArg = (strstr(configArg, "cs_grating2_max=") + 16);
config.cs_grating2_max = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.cs_grating2_max, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing cs_grating2_max\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       cs_grating2_max %s \n", config.cs_grating2_max);
curArg = (strstr(configArg, "cs_write_sleep_grating_change=") + 30);
config.cs_write_sleep_grating_change = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.cs_write_sleep_grating_change, curArg,
       32)) fprintf(logFile, "StripConfigStr: InitializeConfigPointer
       returned false when initializing cs_write_sleep_grating_change\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       cs_write_sleep_grating_change %s \n",
       config.cs_write_sleep_grating_change);
curArg = (strstr(configArg, "cs_write_sleep_filter_change=") + 29);
config.cs_write_sleep_filter_change = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.cs_write_sleep_filter_change, curArg,
       32)) fprintf(logFile, "StripConfigStr: InitializeConfigPointer
       returned false when initializing cs_write_sleep_filter_change\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
cs_write_sleep_filter_change %s \n", config.cs_write_sleep_filter_change);
curArg = (strstr(configArg, "cs_write_sleep_gowave=") + 22);
config.cs_write_sleep_gowave = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.cs_write_sleep_gowave, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing cs_write_sleep_gowave\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       cs_write_sleep_gowave %s \n", config.cs_write_sleep_gowave);
curArg = (strstr(configArg, "cs_filter_1_max=") + 16);
config.cs_filter_1_max = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.cs_filter_1_max, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing cs_filter_1_max\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       cs_filter_1_max %s \n", config.cs_filter_1_max);
curArg = (strstr(configArg, "cs_filter_2_max=") + 16);
config.cs_filter_2_max = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.cs_filter_2_max, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing cs_filter_2_max\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       cs_filter_2_max %s \n", config.cs_filter_2_max);
curArg = (strstr(configArg, "cs_filter_3_max=") + 16);
config.cs_filter_3_max = (char *)calloc(1, 32);
```

```
if (!InitializeConfigPointer(config.cs_filter_3_max, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing cs_filter_3_max\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       cs_filter_3_max %s \n", config.cs_filter_3_max);
curArg = (strstr(configArg, "cs_filter_4_max=") + 16);
config.cs_filter_4_max = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.cs_filter_4_max, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing cs_filter_4_max\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       cs_filter_4_max %s \n", config.cs_filter_4_max);
curArg = (strstr(configArg, "use_ppms_resistivity_bridge=") + 28);
config.use_ppms_resistivity_bridge = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.use_ppms_resistivity_bridge, curArg,
       32)) fprintf(logFile, "StripConfigStr: InitializeConfigPointer
       returned false when initializing use_ppms_resistivity_bridge\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       use_ppms_resistivity_bridge %s \n",
config.use_ppms_resistivity_bridge);
curArg = (strstr(configArg, "rb_rum_readings=") + 16);
config.rb_num_readings = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.rb_num_readings, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing rb_rum_readings\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       rb_rum_readings %s \n", config.rb_num_readings);
curArg = (strstr(configArg, "rb_wait_between_readings=") + 25);
config.rb_wait_between_readings = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.rb_wait_between_readings, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
false when initializing rb_wait_between_readings\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       rb_wait_between_readings %s \n", config.rb_wait_between_readings);
curArg = (strstr(configArg, "rb_read_bridge_1=") + 17);
config.rb_read_bridge_1 = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.rb_read_bridge_1, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing rb_read_bridge_1\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       rb_read_bridge_1 %s \n", config.rb_read_bridge_1);
curArg = (strstr(configArg, "rb_read_bridge_2=") + 17);
config.rb_read_bridge_2 = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.rb_read_bridge_2, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing rb_read_bridge_2\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       rb_read_bridge_2 %s \n", config.rb_read_bridge_2);
curArg = (strstr(configArg, "rb_read_bridge_3=") + 17);
config.rb_read_bridge_3 = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.rb_read_bridge_3, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
              false when initializing rb_read_bridge_3\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       rb_read_bridge_3 %s \n", config.rb_read_bridge_3);
curArg = (strstr(configArg, "rb_num_bridge_read=") + 19);
config.rb_num_bridge_read = (char *)calloc(1, 32);
```

```
if (!InitializeConfigPointer(config.rb_num_bridge_read, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing rb_num_bridge_read\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       rb_num_bridge_read %s \n", config.rb_num_bridge_read);
curArg = (strstr(configArg, "use_SRS830=") + 11);
config.use_SRS830 = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.use_SRS830, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing use_SRS830\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated use_RS830 %s \n",
       config.use_SRS830);
curArg = (strstr(configArg, "SRS830_gpib=") + 12);
config.SRS830_gpib = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.SRS830_gpib, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing SRS830_gpib\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated SRS830_gpib %s
       \n", config.SRS830_gpib);
curArg = (strstr(configArg, "SRS830_chan=") + 12);
config.SRS830_chan = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.SRS830_chan, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing SRS830_chan\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated SRS830_chan %s
       \n", config.SRS830_chan);
curArg = (strstr(configArg, "SRS830_auto_gain=") + 17);
config.SRS830_auto_gain = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.SRS830_auto_gain, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing SRS830_auto_gain\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       SRS830_auto_gain %s \n", config.SRS830_auto_gain);
curArg = (strstr(configArg, "SRS830_write_sleep=") + 19);
config.SRS830_write_sleep = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.SRS830_write_sleep, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing SRS830_write_sleep\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       SRS830_write_sleep %s \n", config.SRS830_write_sleep);
curArg = (strstr(configArg, "use_NOVA=") + 9);
config.use_NOVA = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.use_NOVA, curArg, 32)) fprintf(logFile,
       "StripConfigStr: InitializeConfigPointer returned false when
       initializing use_NOVA\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated use_NOVA %s \n",
       config.use_NOVA);
curArg = (strstr(configArg, "NOVA_meas_sleep=") + 16);
config.NOVA_meas_sleep = (char *)calloc(1, 32);
if (!InitializeConfigPointer(config.NOVA_meas_sleep, curArg, 32))
       fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
       false when initializing NOVA_meas_sleep\n");
fprintf(logFile, "StripConfigStr: This is NULL-terminated
       NOVA_meas_sleep %s \n", config.NOVA_meas_sleep);
curArg = (strstr(configArg, "NOVA_port=") + 10);
config.NOVA_port = (char *)calloc(1, 32);
```

```
	if (!InitializeConfigPointer(config.NOVA_port, curArg, 32)) fprintf(logFile,
		"StripConfigStr: InitializeConfigPointer returned false when
		initializing NOVA_port\n");
	 fprintf(logFile, "StripConfigStr: This is NULL-terminated NOVA_port %s \n",
		config.NOVA_port);
	 curArg = (strstr(configArg, "NOVA_default_write_sleep=") + 25);
	 config.NOVA_default_write_sleep = (char *)calloc(1, 32);
	 if (!InitializeConfigPointer(config.NOVA_default_write_sleep, curArg, 32))
		fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
		false when initializing NOVA_default_write_sleep\n");
	 fprintf(logFile, "StripConfigStr: This is NULL-terminated
		NOVA_default_write_sleep %s \n", config.NOVA_default_write_sleep);
	 curArg = (strstr(configArg, "use_k2400=") + 10);
	 config.use_k2400 = (char *)calloc(1, 32);
	if (!InitializeConfigPointer(config.use_k2400, curArg, 32)) fprintf(logFile,
		"StripConfigStr: InitializeConfigPointer returned false when
		initializing use_k2400\n");
	 fprintf(logFile, "StripConfigStr: This is NULL-terminated use_k2400 %s \n",
		config.use_k2400);
	 curArg = (strstr(configArg, "k2400_meas_sleep=") + 17);
	 config.k2400_meas_sleep = (char *)calloc(1, 32);
	 if (!InitializeConfigPointer(config.k2400_meas_sleep, curArg, 32))
		fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
		false when initializing k2400_meas_sleep\n");
	 fprintf(logFile, "StripConfigStr: This is NULL-terminated
		k2400_meas_sleep %s \n", config.k2400_meas_sleep);
	 curArg = (strstr(configArg, "k2400_port=") + 11);
	 config.k2400_port = (char *)calloc(1, 32);
	 if (!InitializeConfigPointer(config.k2400_port, curArg, 32))
		fprintf(logFile, "StripConfigStr: InitializeConfigPointer returned
		false when initializing k2400_port\n");
	fprintf(logFile, "StripConfigStr: This is NULL-terminated k2400_port %s \n",
		config.k2400_port);
	 return true;
}

bool ReadConfigFile(){
//post-condition: opens config.ini and reads character-by-character into memory
pointed to by configstr.  Calls stripconfigstr with configstr as argument
	 errno_t err;
	 FILE* cfile;
	 if (err = fopen_s(&cfile, "config.ini", "rt") != 0){
		fprintf(logFile, "ReadConfigFile: failed to open config.ini \n");
		return false;
	 }

	 int k = 0;
	 int c = 0;
	 fprintf(logFile, "ReadConfigFile: opened config.ini\n");

	 char *configstr = (char *)malloc(CONFIG_MEM_BLOCK*sizeof(char));
//	 fprintf(logFile, "ReadConfigFile: allocated first memory block\n");
	 c = fgetc(cfile);
//	 fprintf(logFile, "ReadConfigFile: called fgetc(cfile) .ini\n");
	 while (!feof(cfile) && !ferror(cfile)) {
	     configstr[k]=(char)c;
		k++;
		if (k%CONFIG_MEM_BLOCK == 0) {
```

```
                realloc(configstr,
        CONFIG_MEM_BLOCK*sizeof(char)*(k/CONFIG_MEM_BLOCK + 1));
        fprintf(logFile, "ReadConfigFile: allocated another block \n");
        if (configstr == NULL) {
              fprintf(logFile, "ReadConfigFile: Could not expand array to size %i",
                      CONFIG_MEM_BLOCK*sizeof(char)*(k/CONFIG_MEM_BLOCK + 1));
               exit(2);
        }
        }
        c = fgetc(cfile);
        }
        configstr[k] = NULL; // NULL terminate string

        if (!StripConfigString(configstr)){
               fprintf(logFile, "StripConfigFile(configstring) returned false\n");
        }
        if (fclose(cfile)<0) fprintf(logFile, "ReadConfigFile: failed to close
               configfile \n");
        fprintf(logFile, "ReadConfigFile: closed configfile and returning\n");
        return true;
}
```

"CornerStone.cpp"

```
#include "MCControlv9.h"

////////////////////////////////////////////////////////////////////////////
// Cornerstone 260 global variables
////////////////////////////////////////////////////////////////////////////

// Cornerstone 260
bool CS_initialized = false;
int CS_addr = -1;
int CS_dev = -1;
////first wavelength in a sweep
int CS_grating1_startwaveln = -1;
int CS_grating2_startwaveln = -1;
//output wavelength increment size in nm
int CS_stepsize = -1;
//current wavelength
int CS_curwaveln = -1;
//filter currently in place
int CS_curfilter = -1;
// current grating.
int CS_curgrating = -1;
// current shutter 0 = closed; 1 = open
int CS_curshutter = -1;



////////////////////////////////////////////////////////////////////////////

bool shutter (bool openit)
{   //opens and closes the shutter.
       char shutterread[62];
       // new shutterstr holds a\r\n as a string
       // char shutterstr[256] = { 0 };
```

```
        // shutter command to pass as argument to ibwrt
        char cmdstr[256] = { 0 };
        if (openit){
            fprintf(logFile, "CS_shutter: Opening Shutter\n");
            sprintf(cmdstr, "SHUTTER O\n");
        } else{
               fprintf(logFile, "CS_shutter: Closing Shutter\n");
               sprintf(cmdstr, "SHUTTER C\n");
        }
        if (!ibwrite(CS_dev,cmdstr, atoi(config.gpib_write_default_sleep))) {
               fprintf(logFile, "CS_shutter: Could not send SHUTTER command!\n");
               return false;
        }

        sprintf(cmdstr, "SHUTTER?\n");
        if (!ibwrite(CS_dev,cmdstr, atoi(config.gpib_write_default_sleep))) {
               fprintf(logFile, "CS_shutter: Could not send SHUTTER? command!\n");
               return false;
        }
        if (!ibread(CS_dev, shutterread, 62)) {
               fprintf(logFile, "CS_shutter: Could read shutter state!\n");
               return false;
        }

        if (openit) {
               fprintf(logFile, "CS_shutter: Opened Shutter.\n");
               CS_curshutter = 1;
        } else {
               fprintf(logFile, "CS_shutter: Closed Shutter.\n");
               CS_curshutter = 0;
        }
        return true;
}

bool CS_shutter(bool openit) {
        return shutter(openit);
}

// gives command to switch filter and confirms that filter is switched
bool filterswitch(int newfilter) {
        char filterread[62];
        // newfilterstr holds a\r\n as a string
        char newfilterstr[256] = { 0 };
        // filter switch command to pass as argument to ibwrt
        char cmdstr[256] = { 0 };
        fprintf(logFile, "CS_filterswitch: Calling CS_shutter(FALSE).\n");
        CS_shutter(false);
        fprintf(logFile, "CS_filterswitch: Switching from filter %i to
               filter %i.\n", CS_curfilter, newfilter);
        sprintf(cmdstr, "FILTER %i\n", newfilter);
        if (!ibwrite(CS_dev,cmdstr, atoi(config.cs_write_sleep_grating_change))) {
               fprintf(logFile, "CS_filterswitch: Could not send FILTER
                      command!\n");
               return false;
        }
        sprintf(cmdstr, "FILTER?\n");
        if (!ibwrite(CS_dev,cmdstr, atoi(config.gpib_write_default_sleep))) {
```

```
                fprintf(logFile, "CS_filterswitch: Could not send FILTER?
                        command!\n");
                return false;
        }
        if (!ibread(CS_dev, filterread, 62)) {
                fprintf(logFile, "CS_filterswitch: Could not read current filter
                        setting!\n");
                return false;
        }
        sprintf(newfilterstr, "%s\r\n", filterread);
        if (strncmp(filterread, newfilterstr, 1) != 0){
                CS_curfilter = -1;
                fprintf(logFile, "CS_filterswitch: Could not switch filter. Current
                        filter is %s\n", filterread);
                return false;
        }
        fprintf(logFile, "CS_filterswitch: Calling CS_shutter(TRUE).\n");
        CS_shutter(true);
        CS_curfilter = newfilter;
        fprintf(logFile, "CS_filterswitch: Switched to filter %i.\n",
                CS_curfilter);
        return true;
}

// selects the correct filter for a given wavelength and calls filterswitcher if
different than current filter
bool filterchoose(int targetwln) {
        if (targetwln < atoi(config.cs_filter_1_max)){
                if (CS_curfilter != 1) {
                        return filterswitch(1);
                }
        }
        else if (targetwln < atoi(config.cs_filter_2_max)){
                if (CS_curfilter != 2) {
                        return filterswitch(2);
                }
        }
        else if (targetwln < atoi(config.cs_filter_3_max)){
                if (CS_curfilter != 3) {
                        return filterswitch(3);
                }
        }
        else if (targetwln < atoi(config.cs_filter_4_max)){
                if (CS_curfilter != 4) {
                        return filterswitch(4);
                }
        } else {
                if (CS_curfilter != 5) {
                        return filterswitch(5);
                }
    }

        return true;
}


// gives command to switch grating and confirms that grating is switched
bool gratingswitch(int newgrating) {
```

```
char wlread[62]= { 0 };
char gratingread[62] = { 0 };
// curgratingstr holds newgrating\n as a string
char curgratingstr[256] = { 0 };
// grating switch command to pass as argument to ibwrt
char cmdstr[256] = { 0 };
bool reopenshutter = false;

if (CS_curshutter == 1) reopenshutter = true;
if (!shutter(false)) {
	fprintf(logFile, "CS_gratingsswitch: Could not close shutter!\n");
	return false;
}
fprintf(logFile, "CS_gratingsswitch: Switching from grating %i to
	grating %i.\n", CS_curgrating, newgrating);
sprintf(cmdstr, "GRAT %i\n", newgrating);
if (!ibwrite(CS_dev,cmdstr, atoi(config.cs_write_sleep_grating_change))) {
	fprintf(logFile, "CS_gratingsswitch: Could not send GRAT
		command!\n");
	return false;
}

sprintf(cmdstr, "GRAT?\n");
if (!ibwrite(CS_dev,cmdstr, atoi(config.gpib_write_default_sleep))) {
	fprintf(logFile, "CS_gratingsswitch: Could send GRAT? command!\n");
	return false;
}
if (!ibread(CS_dev, gratingread, 62)) {
	fprintf(logFile, "CS_gratingsswitch: Could not read grating
		setting!\n");
	return false;
}

sprintf(curgratingstr, "%i\n", newgrating);
if (strncmp(gratingread, curgratingstr, 1) != 0){
	CS_curgrating = -1;
	fprintf(logFile, "CS_gratingsswitch: Could not switch grating.
		Current grating is %c\n", gratingread[1]);
	return false;
}

CS_curgrating = newgrating;
fprintf(logFile, "CS_gratingsswitch: Switched to grating %i.\n",
	CS_curgrating);

if (reopenshutter) {
	if (!shutter(true)) {
		fprintf(logFile, "CS_gratingsswitch: Could not reopen
			shutter!\n");
		return false;
	}
}

//Deals with CS260 bug where on grating 2 will not accept a gowave command
//until after a wave? one.
sprintf(cmdstr, "WAVE?\n");
if (!ibwrite(CS_dev, cmdstr, atoi(config.gpib_write_default_sleep))) {
```

```
                fprintf(logFile, "CS_gratingswitch: Could not send (bug-fixing)
                        WAVE? command!\n");
                return false;
        }
        if (!ibread(CS_dev, wlread, 62)) {
                fprintf(logFile, "CS_gratingswitch: Could not read current
                        wavelength (for bug-fixing)!\n");
                return false;
        }
        return true;
}

// General goto wavelength command
bool gotowavelength(int targetwln) {
        char wlread[62] = { 0 };
        char cmdstr[256] = { 0 };
        float wlfloat;
        if (CS_curgrating == 1){
                if (targetwln < atoi(config.cs_grating1_start) || targetwln >
                        atoi(config.cs_grating1_max)) {
                        fprintf(logFile, "CS_gotowavelength: Specified wavelength
                        of %i is out of range for grating 1 \n", targetwln);
                        return false;
                }
        }
        if (CS_curgrating == 2){
                if (targetwln < atoi(config.cs_grating2_start) || targetwln >
                        atoi(config.cs_grating2_max)) {
                        fprintf(logFile, "CS_gotowavelength: Specified wavelength
                        of %i is out of range for current grating 2\n", targetwln);
                        return false;
                }
        }
        if (!filterchoose(targetwln)) {
                fprintf(logFile, "CS_gotowavelength: Could not change to appropriate
                        filter!\n");
                return false;
        }
        sprintf(cmdstr, "GOWAVE %i\n", targetwln);
        if (!ibwrite(CS_dev, cmdstr, atoi(config.cs_write_sleep_gowave))) {
                fprintf(logFile, "CS_gotowavelength: Could not send GOWAVE
                        command!\n");
                return false;
        }
        sprintf(cmdstr, "WAVE?\n");
        if (!ibwrite(CS_dev, cmdstr, atoi(config.gpib_write_default_sleep))) {
                fprintf(logFile, "CS_gotowavelength: Could not send WAVE?
                        command!\n");
                return false;
        }
        if (!ibread(CS_dev, wlread, 62)) {
                fprintf(logFile, "CS_gotowavelength: Could not read current
                        wavelength!\n");
                return false;
        }

        wlfloat = (float)atof(wlread);
        CS_curwaveln = round(wlfloat);
```

```
	fprintf(logFile, "CS_gotowavelength: Asked for %i nm, got %f nm (%i nm
		rounded)\n", targetwln, wlfloat, CS_curwaveln);
	if (CS_curwaveln != targetwln) { fprintf(logFile, "CS_gotowavelength:
		Target and actual wavelength do not match!\n"); return false; }
	return true;
}

//Cornerstone 260 does not have internal initialization routine, purpose of
CS_init() parse command line and check communications
bool CS_init() {

	if (CS_initialized == true) return true;
	fprintf(logFile, "CS_init: Beginning Initialization.\n");
	CS_addr = atoi(config.cs_gpib);
	if ((CS_addr < 1) || (CS_addr > 30)) {
		fprintf(logFile, "CS_init: No or Invalid GPIB address %i
			specified!\n", CS_addr);
		return false;
	}
	CS_grating1_startwaveln = atoi(config.cs_grating1_start);
	CS_grating2_startwaveln = atoi(config.cs_grating2_start);
	CS_stepsize = atoi(config.cs_stepsize);
	if (CS_stepsize == 0) {
		fprintf(logFile, "CS_init: Invalid wavelength step of %i
		specified.\n", CS_stepsize);
		return false;
	}

	fprintf(logFile, "CS_init: GPIBaddress  = %i.\n", CS_addr);
	fprintf(logFile, "CS_init: Lambda-Grating1 Start = %i.\n",
		CS_grating1_startwaveln);
	fprintf(logFile, "CS_init: Lambda-Grating2 Start = %i.\n",
		CS_grating2_startwaveln);
	fprintf(logFile, "CS_init: Lambda-Step  = %i.\n", CS_stepsize);
	CS_dev = ibdevice(CS_addr, 0 /* No Secondary Address */, 13 /* 30 sec
		timeout */, 0 /* Enable END */, GPIB_READ_TERM_CHAR /* End of string
		code */);
	if (CS_dev < 1) {
		fprintf(logFile, "CS_init: Could not open communications with
		device!\n");
		return false;
	}

	// At this point we are initialized and can accept commands through CS_do
	CS_initialized = true;
	// go to user specified starting wavelength and correct grating
	fprintf(logFile, "CS_init: Closing shutter and going to starting
		wavelength\n");
	if (!CS_do(CS_ADVISE_CLOSESHUTTER)) {
		fprintf(logFile, "CS_init: Could not close shutter!\n");
		CS_initialized = false;
		return false;
	}
	if (!CS_do(CS_ADVISE_RESET_GRATING_1)) {
		fprintf(logFile, "CS_init: Could not go to starting wavelength!\n");
		CS_initialized = false;
		return false;
	}
```

```
        fprintf(logFile, "CS_init: Ready to accept commands.\n");
        return true;
}

// Handle all the necessary commands to control the monochromator
bool CS_do(int advise) {
        if (CS_initialized != true) {
            fprintf(logFile, "CS_do: Monochromator not initialized!\n");
            return false;
        }

        switch (advise) {
            case CS_ADVISE_RESET_GRATING_1: //go to start scan wavelength
                        fprintf(logFile, "CS_do: Resetting to grating 1\n");
                        if (CS_curgrating == 1) fprintf(logFile, "CS_do: Already using
                                grating 1 \n");
                        else {
                                if (!gratingswitch(1)){
                                        fprintf(logFile, "CS_do: Could not reset to
                                        grating 1\n");
                                CS_curgrating = -1;
                                return false;
                                }
                        }
                        fprintf(logFile, "CS_do: Going to scan start wavelength %i\n",
                                CS_grating1_startwaveln);
                        if (!gotowavelength(CS_grating1_startwaveln)) {
                                fprintf(logFile, "CS_do: Error returning to grating
                                        1start wavelength!\n");
                                return false;
                        } else {
                                CS_curgrating = 1;
                                fprintf(logFile, "CS_do: Returned to wavelength %i\n",
                                        CS_curwaveln);
                        }
                        break;

        case CS_ADVISE_RESET_GRATING_2: //go to start scan wavelength
                fprintf(logFile, "CS_do: Resetting to grating 2\n");
                if (CS_curgrating == 2) fprintf(logFile, "CS_do: Already using
                        grating 2 \n");
                else {
                        if (!gratingswitch(2)){
                                fprintf(logFile, "CS_do: Could not reset to
                                        grating 2\n");
                                CS_curgrating = -1;
                                return false;
                                }
                        }
                        fprintf(logFile, "CS_do: Going to scan start wavelength %i\n",
                                CS_grating2_startwaveln);
                        if (!gotowavelength(CS_grating2_startwaveln)) {
                                fprintf(logFile, "CS_do: Error returning to grating
                                        2start wavelength!\n");
                        CS_curgrating = -1;
                        return false;
                        } else {
```

```
                        fprintf(logFile, "CS_do: Returned to wavelength %i\n",
                                CS_curwaveln);
                }
                break;

        case CS_ADVISE_INCRWL: //increment wavelength
                fprintf(logFile, "CS_do: Incrementing wavelength to %i\n",
                (CS_curwaveln + CS_stepsize));
                if (!gotowavelength(CS_curwaveln + CS_stepsize)) {
                        fprintf(logFile, "CS_do: Error incrementing
                                wavelength!\n");
                return false;
                } else {
                        fprintf(logFile, "CS_do: Incremented wavelength
                                to %i\n", CS_curwaveln);
                }
                break;

        case CS_ADVISE_CLOSESHUTTER:
                fprintf(logFile, "CS_do: Closing shutter\n");
                if (!shutter(false)) {
                        fprintf(logFile, "CS_do: Error closing shutter!\n");
                        return false;
                } else {
                        fprintf(logFile, "CS_do: Shutter closed.\n");
                }
                break;

        case CS_ADVISE_OPENSHUTTER:
                fprintf(logFile, "CS_do: Opening shutter\n");
                if (!shutter(true)) {
                        fprintf(logFile, "CS_do: Error opening shutter!\n");
                        return false;
                } else {
                        fprintf(logFile, "CS_do: Shutter opened.\n");
                }
                break;

        default:
                fprintf(logFile, "CS_do: Illegal Option of %i", advise);
                return false;
        }

        return true;
}
```

"k2400.cpp"

```
#include "MCControlv9.h"

////////////////////////////////////////////////////////////////////////////
// K2400 Source Meter global variables
////////////////////////////////////////////////////////////////////////////

//points to serial port
HANDLE k2400Serial;
bool k2400_initialized = false;
int k2400_measure_sleep = -1;
```

```
double k2400_last_value = 0.0;

////////////////////////////////////////////////////////////////////////////

//function PS_read_lamp_state() exists even though the command START\r
automatically returns the lamp state, because if you attempt to start when the
lamp is already on or stop it when the lamp is already off, you get an error



bool k2400_measure(){
	char cmdstr[256] = { 0 };
	char readstr[256] = { 0 };
	Sleep(k2400_measure_sleep);
	DWORD dwBytesRead = 0;

	sprintf(cmdstr, ":TRAC:CLE\r");
	fprintf(logFile, "k2400_measure: command string%s\n", cmdstr);
	if(!WriteFile(k2400Serial, cmdstr, 10, &dwBytesRead, NULL)){
		fprintf(logFile, "k2400_measure: error writing command :TRAC:CLE
			\n");//error occurred. Report to user.
		return false;
	}
	sprintf(cmdstr, "MEAS:VOLT?\r");
	fprintf(logFile, "k2400_measure: command string%s\n", cmdstr);
	if(!WriteFile(k2400Serial, cmdstr, 11, &dwBytesRead, NULL)){
		fprintf(logFile, "k2400_measure: error writing command MEAS:VOLT?
			\n");//error occurred. Report to user.
		return false;
	}
	Sleep(100);
	if(!ReadFile(k2400Serial, readstr, 256, &dwBytesRead, NULL)){
		fprintf(logFile, "k2400_measure: error reading k2400
			voltage\n");//error occurred. Report to user.
		return false;
		}
	k2400_last_value = atof(readstr);
	fprintf(logFile, "k2400_measure: read %e V\n", k2400_last_value);
	k2400_last_value = atof(readstr);
	return true;
}

bool k2400_init(){
	k2400_measure_sleep = atoi(config.k2400_meas_sleep);
      k2400Serial = CreateFile(config.k2400_port, GENERIC_READ | GENERIC_WRITE, 0,
		0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);

	if(k2400Serial==INVALID_HANDLE_VALUE){
		if(GetLastError()==ERROR_FILE_NOT_FOUND){
			fprintf(logFile, "k2400_init: serial port does not exist %s
				\n", config.k2400_port);
			return false;
		}
	fprintf(logFile, "k2400_init: serial port exists, but invalid handle value
		\n");
	}
	DCB dcbSerialParams = {0};
	dcbSerialParams.DCBlength=sizeof(dcbSerialParams);
```

```
	if (!GetCommState(k2400Serial, &dcbSerialParams)) {
		fprintf(logFile, "k2400_init: error getting serial port state \n");
	}
	dcbSerialParams.BaudRate=CBR_9600;
	dcbSerialParams.ByteSize=8;
	dcbSerialParams.StopBits=ONESTOPBIT;
	dcbSerialParams.Parity=NOPARITY;
	if(!SetCommState(k2400Serial, &dcbSerialParams)){
		fprintf(logFile, "k2400_init: error setting serial port state \n");
	}
	//set serial write and read timeouts
	//PC02202012 hard-coding the timeout values because I can't forsee a
	circumstance in which it would be valuable to change them on the fly
	COMMTIMEOUTS timeouts={0};
	timeouts.ReadIntervalTimeout=50;
	timeouts.ReadTotalTimeoutConstant=50;
	timeouts.ReadTotalTimeoutMultiplier=10;
	timeouts.WriteTotalTimeoutConstant=50;
	timeouts.WriteTotalTimeoutMultiplier=10;
	if(!SetCommTimeouts(k2400Serial, &timeouts)){
		fprintf(logFile, "k2400_init: unknown error \n");
	}
	//clear out initial value in read stack
	DWORD initBytesRead = 0;
	char initreadstr[256] = { 0 };
	if(!ReadFile(k2400Serial, initreadstr, 256, &initBytesRead, NULL)){
		fprintf(logFile, "k2400_init: can't read from %s \n",
			config.k2400_port);
		return false;
	}
	char cmdstr[256] = { 0 };
	DWORD dwBytesRead = 0;
	sprintf(cmdstr, "REN\r");
	if(!WriteFile(k2400Serial, cmdstr, 4, &dwBytesRead, NULL)){
		fprintf(logFile, "k2400_init: error setting remote operation
			\n");//error occurred. Report to user.
		return false;
	}
	fprintf(logFile, "k2400_init: k2400 successfully initialized \n");
	k2400_initialized = true;
	return true;
}

bool k2400_do(int advise){
	if (k2400_initialized != true) {
	    fprintf(logFile, "k2400_do: k2400 not initialized!\n");
	    return false;
	}
	switch (advise) {
	    case k2400_MEASURE:
			fprintf(logFile, "k2400_do: calling k2400_measure \n");
			if (!k2400_measure()) {
				fprintf(logFile, "k2400_do: Error measuring \n");
				return false;
			}
			break;
```

```
            default:
                        fprintf(logFile, "k2400_init: Illegal Option of %i", advise);
                        return false;
        }
        return true;
}
```

"MCControl9.cpp"

```
#include "MCControlv9.h"

////////////////////////////////////////////////////////////////////////////
//Common global variables
////////////////////////////////////////////////////////////////////////////

// Log File
FILE *logFile = NULL;

// Data File
FILE *dataFile = NULL;


// Session identifier
int MC_uid = 0;

// Configuration
configuration config;

////////////////////////////////////////////////////////////////////////////

LRESULT CALLBACK PPMSAdviseClassReceiver(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam);
void PPMSProcessAdvise(HWND hWnd, WPARAM wParam, LPARAM lParam);


int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
    // Open Log File
       logFile = fopen(MC_LOG_FILE, "a");
       if (logFile == NULL)
              return 1; // log file cannot be opened for writing

       dataFile = fopen(MC_DATA_FILE, "a");
       if (dataFile == NULL)
              return 2;


    // Generate unique ID for this session
       srand( (unsigned)time( NULL ) );
       MC_uid = rand();


       // Output initial line to log file
```

```
        fprintf(logFile, "MCControl Started at %i with MC_uid=%i\n", time(NULL),
                MC_uid);

        // Make both files bufferless (in case there are crashes)
        if (setvbuf(dataFile, NULL, _IONBF, 0) != 0 ||
                setvbuf(logFile, NULL, _IONBF, 0) != 0) {
                        fprintf(logFile, "MCControl: FATAL ERROR: CANNOT FORCE FILE
                                ACCESS TO BE BUFFERLESS.\n");
                return 3; // file streams cannot be bufferless
        }

        //Read config.ini
        fprintf(logFile, "MCControl: before readconfig\n");
        ReadConfigFile();
        fprintf(logFile, "MCControl: after readconfig\n");
        // Write GPIB Board Configuration and data file header
        WriteGPIBBoardConfig();
        PPMSWriteDataHeader();

if (strncmp(strupr(config.use_ppms), "FALSE", 5) == 0){
        // read_sequence_file
        if (!ReadSeqFile()){
                fprintf(logFile, "MCControl: could not load sequence\n");
                return 4;
        }
}


if (strncmp(strupr(config.use_power_supply), "TRUE", 4) == 0){
        // Initialize power supply
        if (!PS_init()){
                fprintf(logFile, "MCControl: Power Supply failed to initialize\n");
                return 5;
        }
}

if (strncmp(strupr(config.use_NOVA), "TRUE", 4) == 0){
        // Initialize power supply
        if (!NOVA_init()){
                fprintf(logFile, "MCControl: NOVA failed to initialize\n");
                return 6;
        }
}

if (strncmp(strupr(config.use_k2400), "TRUE", 4) == 0){
        // Initialize power supply
        if (!k2400_init()){
                fprintf(logFile, "MCControl: k2400 failed to initialize\n");
                return 7;
        }
}

if (strncmp(strupr(config.use_cornerstone260), "TRUE", 4) == 0){
        // Initialize monochromator
        if (!CS_init()){
                fprintf(logFile, "MCControl: Cornerstone 260 Monochromator failed to
                        initialize\n");
                return 8;
```

```
        }
}


if (strncmp(strupr(config.use_SRS830), "TRUE", 4) == 0){
        // Initialize SRS830
        if (!init_SRS830()){
                fprintf(logFile, "MCControl: SRS830 failed to initialize\n");
                return 9;
        }
}



        // Create window to receive PPMS events and show it
        // (also acts as a log of what happens)
        WNDCLASSEX PPMSAdviseClass;
        PPMSAdviseClass.cbSize = sizeof(PPMSAdviseClass);
        PPMSAdviseClass.style = CS_HREDRAW | CS_VREDRAW;
        PPMSAdviseClass.lpfnWndProc = (WNDPROC)PPMSAdviseClassReceiver;
        PPMSAdviseClass.cbClsExtra= 0;
        PPMSAdviseClass.cbWndExtra= 0;
        PPMSAdviseClass.hInstance = hInstance;
        PPMSAdviseClass.hIcon = 0;
        PPMSAdviseClass.hIconSm = 0;
        PPMSAdviseClass.hCursor = 0;
        PPMSAdviseClass.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
        PPMSAdviseClass.lpszMenuName = 0;
        PPMSAdviseClass.lpszClassName =  _T("PPMSAdviseClass");
        RegisterClassEx(&PPMSAdviseClass);

        HWND PPMSAdviseWindow = CreateWindow(_T("PPMSAdviseClass"), _T("PPMS
                Monochromator Control"),
                WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, 100, 50, NULL,
                NULL, hInstance, NULL);

        ShowWindow(PPMSAdviseWindow, nCmdShow);
        UpdateWindow(PPMSAdviseWindow);

        // Register Window to receive advisories
        HANDLE AdvisoriesRegister = RegisterForAdvisories(PPMSAdviseWindow,
                PPMS_ADVISORY, 0);
        int k = 0;
        LPARAM NULLParam = NULL;
        // process go back advises.
        if (strncmp(strupr(config.use_ppms), "FALSE", 4) == 0){
                while (seqlist[k] != NULL){
                        fprintf(logFile, "MCControl: inside while loop, k = %i,
                                seqlist[k] = %ld.\n", k, seqlist[k]);
                        if ((seqlist[k] > (int)GO_BACK_MIN_ADVISE) && (seqlist[k] <
                                (int)GO_BACK_MAX_ADVISE)){
                                seqlist[k] += -100;
                                k += (-(seqlist[k]%100));
                        }
                        PPMSProcessAdvise(PPMSAdviseWindow, seqlist[k], NULLParam);
                        if ((seqlist[k] > (GO_BACK_MIN_ADVISE - 100)) && (seqlist[k] <
                                GO_BACK_MIN_ADVISE)){
                                seqlist[k] = seqlist_original[k];
```

```
                }
                k++;
            }
        }

        // Infinite Message Loop
        fprintf(logFile, "MCControl: Waiting for Advises.\n");
        MSG msg;
        while (GetMessage(&msg, NULL, 0, 0)) {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }

        // Unregister Window to stop receiving advisories
        UnregisterFromAdvisories( AdvisoriesRegister );
        CloseHandle(hSerial);
        CloseHandle(k2400Serial);
        CloseHandle(NovaSerial);
        // Close log and data files
        fprintf(logFile, "MCControl: Ended at %i for uid=%i\n", time(NULL),
            MC_uid);
        fclose(logFile);
        fclose(dataFile);
        return 0;
}

LRESULT CALLBACK PPMSAdviseClassReceiver(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam) {
        int k = 0;

        switch (message) {
        case WM_COMMAND:
            // handle menu selections etc.
        break;
        case WM_PAINT:
            // draw our window - note: must do something here
        break;
        case WM_DESTROY:
            PostQuitMessage(0);
                break;
           case PPMS_ADVISORY:
            fprintf(logFile, "MCControl: Got PPMS Advisory %i\n", wParam);
               PPMSProcessAdvise(hWnd, wParam, lParam);
                  break;
        default:
            // We do not want to handle this message so pass back to Windows
            // to handle it in a default way
            return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}


// Process PPMS advises. wParam contains the advise number from the PPMS Multivu
sequence
// The values of wParam (range: 1-1999) used for this program are given in the
relevent
// header files.
```

```
void PPMSProcessAdvise(HWND hWnd, WPARAM wParam, LPARAM lParam) {
	char response[4096];		// response buffer
	long responseSize=4096;	// sizeof response buffer

	//pause the 6000 sequence until done processing (timeout = 1440 min)
	if (strncmp(strupr(config.use_ppms), "TRUE", 4) == 0){
		PPMSSendAndReceive("HOLDOFF 1 1440", response, responseSize);
	}

	if (wParam >= CS_MIN_ADVISE && wParam <= CS_MAX_ADVISE){
		if (strncmp(strupr(config.use_cornerstone260), "TRUE", 4) == 0)
			CS_do((int)wParam);
		if (strncmp(strupr(config.use_cornerstone260), "FALSE", 4) == 0){
			 fprintf(logFile, "PPMSProcessAdvise: Got Advisory %i, but not
				using Cornerstone260 /n", wParam);
		}
	}

	if (wParam == ADVISORY_WRITE_MEASUREMENT){
		if (strncmp(strupr(config.use_SRS830), "TRUE", 4) == 0)
	SRS830_meas();
		PPMSWriteMeasurement();
	}
	if (wParam >= PS_MIN_ADVISE && wParam <= PS_MAX_ADVISE){
		if (strncmp(strupr(config.use_power_supply), "TRUE", 4) == 0)
			PS_do((int)wParam);
		if (strncmp(strupr(config.use_power_supply), "FALSE", 4) == 0){
			fprintf(logFile, "PPMSProcessAdvise: Got Advisory %i, but not
				using PowerSupply /n", wParam);
		}

	}
	if (wParam >= k2400_MIN_ADVISE && wParam <= k2400_MAX_ADVISE){
		if (strncmp(strupr(config.use_k2400), "TRUE", 4) == 0)
			k2400_do((int)wParam);
		if (strncmp(strupr(config.use_k2400), "FALSE", 4) == 0){
			fprintf(logFile, "PPMSProcessAdvise: Got Advisory %i, but not
			using k2400 /n", wParam);
		}

	}
	if (wParam >= NOVA_MIN_ADVISE && wParam <= NOVA_MAX_ADVISE){
		if (strncmp(strupr(config.use_NOVA), "TRUE", 4) == 0)
			NOVA_do((int)wParam);
		if (strncmp(strupr(config.use_NOVA), "FALSE", 4) == 0){
			fprintf(logFile, "PPMSProcessAdvise: Got Advisory %i, but not
			using NOVA /n", wParam);
		}
	}
	if (strncmp(strupr(config.use_ppms), "FALSE", 4) == 0 && wParam ==
		GENERAL_WAIT_ADVISE){
		Sleep(atoi(config.general_wait_time));
		fprintf(logFile, "PPMSProcessAdvise: sleeping for %i ms /n",
		atoi(config.general_wait_time));
	}
	// resume the 6000 sequence
	if (strncmp(strupr(config.use_ppms), "TRUE", 4) == 0){
```

```
                PPMSSendAndReceive("HOLDOFF -1", response, responseSize);
        }
        return;
}

"Powersupply.cpp"

#include "MCControlv9.h"
#include <stdio.h>
#include <wchar.h>

////////////////////////////////////////////////////////////////////////////
// 300W Power Supply global variables
////////////////////////////////////////////////////////////////////////////

//points to serial port
HANDLE hSerial;
bool PS_initialized = false;
//first power in a sweep
int PS_initial_power = -1;
//power increment size in Watts
int PS_stepsize = -1;
//current power
int PS_curpower = -1;
// is the lamp on or off?  FALSE for off, TRUE for on
bool PS_lamp_state = FALSE;
int PS_default_write_sleep = -1;

////////////////////////////////////////////////////////////////////////////

//function PS_read_lamp_state() exists even though the command START\r
automatically returns the lamp state, because if you attempt to start when the
lamp is already on or stop it when the lamp is already off, you get an error

bool PS_read_lamp_state(){
        char cmdstr[256] = { 0 };
        char readstr[256] = { 0 };
        DWORD dwBytesRead = 0;
        sprintf(cmdstr, "STB?\r");
        if(!WriteFile(hSerial, cmdstr, 5, &dwBytesRead, NULL)){
                fprintf(logFile, "PS_read_lamp_state: error writing command ESR?
                \n");//error occurred. Report to user.
                return false;
        }
        Sleep(PS_default_write_sleep);
        if(!ReadFile(hSerial, readstr, 256, &dwBytesRead, NULL)){
                fprintf(logFile, "PS_read_lamp_state: error reading lamp state
                        \n");//error occurred. Report to user.
                        return false;
        }
        fprintf(logFile, "PS_read_lamp_state: readstr %s \n", readstr);
        if (!(strncmp(readstr, "STB21", 5) == 0) && !(strncmp(readstr, "STBA1", 5)
                == 0)){
                fprintf(logFile, "PS_read_lamp_state: unknown error %s \n",
                        readstr);
                return false;
        }
        if ((strncmp(readstr, "STB21", 5) == 0)){
```

```
                fprintf(logFile, "PS_read_lamp_state: lamp is currently off \n");
                PS_lamp_state = FALSE;
                return true;
        }
        if ((strncmp(readstr, "STBA1", 5) == 0)){
                fprintf(logFile, "PS_read_lamp_state: lamp is currently on \n");
                PS_lamp_state = TRUE;
                return true;
        }
        return false;
}

bool PS_change_lamp_state(bool newstate){
        if (PS_lamp_state == newstate){
                fprintf(logFile, "PS_change_lamp_state: lamp already ");
                if (PS_lamp_state){
                        fprintf(logFile, "ON \n");
                    return TRUE;
                }
                fprintf(logFile, "OFF \n");
                return FALSE;
        }
        char cmdstr[256] = { 0 };
        char readstr[256] = { 0 };
        DWORD dwBytesRead = 0;
        if (newstate){
                sprintf(cmdstr, "START\r");
                if(!WriteFile(hSerial, cmdstr, 6, &dwBytesRead, NULL)){
                        fprintf(logFile, "PS_change_lamp_state: error writing command
                                START \n");//error occurred. Report to user.
                }
        }
        if (!newstate){
                fprintf(logFile, "PS_change_lamp_state: writing stop \n");
                sprintf(cmdstr, "STOP\r");
                if(!WriteFile(hSerial, cmdstr, 5, &dwBytesRead, NULL)){
                        fprintf(logFile, "PS_change_lamp_state: error writing command
                                STOP \n");//error occurred. Report to user.
                }
        }
        Sleep(PS_default_write_sleep);
        if(!ReadFile(hSerial, readstr, 256, &dwBytesRead, NULL)){
                                fprintf(logFile, "PS_change_lamp_state: error reading
                                        serial port power \n");//error occurred. Report
                                        to user.
        }
        fprintf(logFile, "PS_change_lamp_state: readstr: %s\n", readstr);
        PS_read_lamp_state();
        if (PS_lamp_state == newstate){
                fprintf(logFile, "PS_lamp_on_or_off: successfully changed lamp state
                        to %s \n",(PS_lamp_state)?"on":"off");
                return true;
        }
        return false;
}

bool PS_change_power(int power){
        if ((40 > power) || (power > 100)){
```

```
		fprintf(logFile, "PS_change_power: invalid value for power \n");
		return false;
	}
	char cmdstr[256] = { 0 };
	char readstr[256] = { 0 };
	fprintf(logFile, "PS_change_power: power: %i\n", power);
	sprintf(cmdstr, "P-PRESET=00%i\r", power);
	fprintf(logFile, "PS_change_power: cmdstr %s\n", cmdstr);
	DWORD dwBytesRead = 0;
	if(!WriteFile(hSerial, cmdstr, 14, &dwBytesRead, NULL)){
		fprintf(logFile, "PS_change_power: error writing command P-PRESET
			\n");//error occurred. Report to user.
	}
	Sleep(PS_default_write_sleep);
	if(!ReadFile(hSerial, readstr, 256, &dwBytesRead, NULL)){
		fprintf(logFile, "PS_change_power: error reading serial
		port power \n");//error occurred. Report to user.
	}
	fprintf(logFile, "PS_change_power: readstr: %s\n", readstr);
	sprintf(cmdstr, "P-PRESET?\r", power);
	if(!WriteFile(hSerial, cmdstr, 10, &dwBytesRead, NULL)){
		fprintf(logFile, "PS_change_power: error writing command P-PRESET?
		\n");//error occurred. Report to user.
	}
	if(!ReadFile(hSerial, readstr, 256, &dwBytesRead, NULL)){
		fprintf(logFile, "PS_change_power: error reading serial       port
		\n");//error occurred. Report to user.
	}
	fprintf(logFile, "PS_change_power: readstr: %s\n", readstr);
	int powint = atoi(readstr);
	fprintf(logFile, "PS_change_power: powint: %i\n", powint);
	if (powint != power){
		fprintf(logFile, "PS_change_power: returned power not the same as
			sent power \n");
	}
	PS_curpower = powint;
	return true;
}



bool PS_init(){
	PS_initial_power=atoi(config.ps_initial_power);
	PS_stepsize=atoi(config.ps_stepsize);
	PS_default_write_sleep=atoi(config.ps_default_write_sleep);
	if ((PS_initial_power < 40) || (PS_initial_power > 100)) {
		fprintf(logFile, "PS_init: Invalid initial power specified %i\n",
			PS_initial_power);
		return false;
	}
	if ((PS_stepsize < 1) || (PS_stepsize > 60)) {
		fprintf(logFile, "PS_init: Invalid stepsize specified %i\n",
		PS_stepsize);
		return false;
	}
	PS_curpower = 0;
	fprintf(logFile, "PS_init: this is wport %s \n", config.ps_port);
```

```
hSerial = CreateFile(config.ps_port, GENERIC_READ | GENERIC_WRITE, 0, 0,
	OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
if(hSerial==INVALID_HANDLE_VALUE){
	if(GetLastError()==ERROR_FILE_NOT_FOUND){
		fprintf(logFile, "PS_init: serial port does not exist %s \n",
			config.ps_port);
		return false;
	}
	fprintf(logFile, "PS_init: serial port exists, but invalid handle
		value \n");
}
DCB dcbSerialParams = {0};
dcbSerialParams.DCBlength=sizeof(dcbSerialParams);
if (!GetCommState(hSerial, &dcbSerialParams)) {
	fprintf(logFile, "PS_init: error getting serial port state \n");
}
dcbSerialParams.BaudRate=CBR_9600;
dcbSerialParams.ByteSize=8;
dcbSerialParams.StopBits=ONESTOPBIT;
dcbSerialParams.Parity=NOPARITY;
if(!SetCommState(hSerial, &dcbSerialParams)){
	fprintf(logFile, "PS_init: error setting serial port state \n");
}
//set serial write and read timeouts
//PC02202012 hard-coding the timeout values because I can't forsee a
circumstance in which it would be valuable to change them on the fly
COMMTIMEOUTS timeouts={0};
timeouts.ReadIntervalTimeout=50;
timeouts.ReadTotalTimeoutConstant=50;
timeouts.ReadTotalTimeoutMultiplier=10;
timeouts.WriteTotalTimeoutConstant=50;
timeouts.WriteTotalTimeoutMultiplier=10;
if(!SetCommTimeouts(hSerial, &timeouts)){
	fprintf(logFile, "PS_init: unknown error \n");
}
//clear out initial value in read stack
DWORD initBytesRead = 0;
char initreadstr[256] = { 0 };
if(!ReadFile(hSerial, initreadstr, 256, &initBytesRead, NULL)){
	fprintf(logFile, "PS_init: can't read from %s \n", config.ps_port);
}
//error register reads error if you attempt to turn lamp on when it's
already on or off when it's already off
if(!PS_read_lamp_state()) fprintf(logFile, "PS_init: PS_read_lamp_state
	returned false \n");
	fprintf(logFile, "PS_init: PS_read_lamp_state is %s
	\n",(PS_lamp_state)?"true":"false");
if(!PS_lamp_state) {
	fprintf(logFile, "PS_init: lamp was off when PS_init() started\n");
	if(!PS_change_lamp_state(TRUE)){
		return false;
	}
}
if(!PS_change_power(PS_initial_power)){
	fprintf(logFile, "PS_init: change_power(config.ps_initial_power)
		returned false \n");
	return false;
}
```

```
        fprintf(logFile, "PS_init: power supply successfully initiated \n");
        PS_initialized = true;
        return true;
}

bool PS_do(int advise){
        if (PS_initialized != true) {
            fprintf(logFile, "PS_do: power supply not initialized!\n");
            return false;
        }
        switch (advise) {
            case PS_ADVISE_RESETPOW: //go to start scan wavelength
                    fprintf(logFile, "PS_do: Going to scan start power %i\n",
                            PS_initial_power);
                    if (!PS_change_power(PS_initial_power)) {
                            fprintf(logFile, "PS_do: Error returning to intial
                                    power!\n");
                            return false;
                    } else {
                            PS_curpower = PS_initial_power;
                            fprintf(logFile, "PS_do: Returned to start scan power
                                    \n");
                    }
                    break;

            case PS_ADVISE_INCRPOW: //increment wavelength
                    fprintf(logFile, "PS_do: Incrementing power to %i\n",
                            (PS_curpower + PS_stepsize));

                    if (!PS_change_power(PS_curpower + PS_stepsize)) {
                            fprintf(logFile, "PS_do: Error incrementing power!\n");
                            return false;
                    } else {
                            fprintf(logFile, "PS_do: Incremented power to %i\n",
                                    PS_curpower);
                    }
                    break;

               case PS_ADVISE_LAMP_OFF: // turn lamp off
                    fprintf(logFile, "PS_do: Turning lamp off\n");

                    if (!PS_change_lamp_state(FALSE)) {
                            fprintf(logFile, "PS_do: Error turning lamp off!\n");
                            return false;
                    } else {
                            fprintf(logFile, "PS_do: Lamp turned off %i\n");
                    }
                    break;

               case PS_ADVISE_LAMP_ON: // turn lamp of
                    fprintf(logFile, "PS_do: Turning lamp on\n");

                    if (!PS_change_lamp_state(TRUE)) {
                            fprintf(logFile, "PS_do: Error turning lamp on!\n");
                            return false;
                    } else {
                            fprintf(logFile, "PS_do: Lamp turned on %i\n");
                    }
```

```
                    break;

            default:
                    fprintf(logFile, "PS_do: Illegal Option of %i", advise);
                    return false;
        }
        return true;
}
```

"ppmsutil.cpp"

```
#include "MCControlv9.h"
#include <math.h>

// Talk to the Model 6000 of the PPMS
long PPMSSendAndReceive(char *cmd, char *response, int responseSize) {
        long bytesRead;            // pointer to actual bytes read
        int pPpmsError;            // command error
        char errorStr[256];        // error string
        long errorSize=256;        // sizeof error string

        GpibSend( atoi(config.ppms_gpib_address), cmd, (long)strlen(cmd), response,
                responseSize, &bytesRead, &pPpmsError, errorStr, errorSize, 0 /*
                Don't swap bytes */, atoi(config.gpib_comm_mutex_timeout) );
                return bytesRead;
}

// Run GPIB command after getting access to the bus through the PPMS hardware
bool ibwrite(int ud, char *cmd, int sleep) {
        int writeStatus, writeError;
        if (strlen(cmd) > 0) {
                // Get Bus
                GetCommunicationMutex(atoi(config.gpib_comm_mutex_timeout));

                // Run Command
                ibwrt(ud, cmd, (long)strlen(cmd));
                writeStatus = ibsta;
                writeError = iberr;

                // Release Bus
                ReleaseCommunicationMutex();

                if (writeStatus & ERR){
                        fprintf(logFile, "ibwrite: Could not execute
                                ibwrite(%i, %s, %i) with error ibsta=%i and
                                iberr=%i.\n", ud, cmd, sleep, writeStatus, writeError);
                        return false;
                }

                // Sleep
                if (sleep > 0) Sleep(sleep);
                } else {
                fprintf(logFile, "ibwrite: Null command passed. Executing
                        nothing.\n");
                return false;
    }
        return true;
}
```

```
int ibdevice(int pad, int sad, int tmo, int eot, int eos) {
	int rv;
	GetCommunicationMutex(atoi(config.gpib_comm_mutex_timeout));
	rv = ibdev(atoi(config.gpib_board), pad, sad, tmo, eot, eos);
	if (ibsta & ERR){
		fprintf(logFile, "ibdevice: Could not execute ibdevice with error
			ibsta=%i and iberr=%i.\n", ibsta, iberr);
		rv = 0;
	}

	ReleaseCommunicationMutex();
	return rv;
}

// Read from GPIB after getting access to the bus through the PPMS hardware
bool ibread(int ud, char *s, long cnt) {
	int count = 0, scnt = 0;
	char c = 0;
	int readStatus, readError;

	// Get Bus
	GetCommunicationMutex(atoi(config.gpib_comm_mutex_timeout));

	// Read until there is nothing left
	ibrd(ud, &c, 1);
	while (c != GPIB_READ_TERM_CHAR && iberr == 0) {
		count++;
		if (scnt < cnt-2) {
			s[scnt] = c; scnt++;
		}
		ibrd(ud, &c, 1);
	}
	readStatus = ibsta;
	readError = iberr;

	// Release Bus
	ReleaseCommunicationMutex();

	// Add GPIB_READ_TERM_CHAR to string
	if (c == GPIB_READ_TERM_CHAR) { s[scnt] = GPIB_READ_TERM_CHAR; scnt++; }
	// Null terminate
	s[scnt] = 0; scnt++;

	if (readStatus & ERR) {
		fprintf(logFile, "ibread: Could not execute ibread with error
			ibsta=%i and iberr=%i.\n", readStatus, readError);
		return false;
	}

	// If more bytes are read than requested, that is an error
	if (count > cnt-1) {
		fprintf(logFile, "ibread: Device %i returned %i bytes when only (%i-
			1) were requested!\n", ud, count, cnt);
		return false;
	}
	// If more bytes are requested than read, that is NOT an error
	//fprintf(logFile, "ibread: Device %i returned %s \n", ud, s);
```

```
	// A non-zero iberr is a warning
	if (iberr != 0)
		fprintf(logFile, "ibread: Warning, device %i returned iberr=%i!\n",
			ud, iberr);
	return true;
}

void WriteGPIBBoardConfig() {
	int temp;
	// Record GPIB board configuration (for debugging purposes)
	fprintf(logFile, " GPIB Board Configuration:\n");
	ibask(atoi(config.gpib_board), IbaPAD, &temp);
	fprintf(logFile, "  IbaPAD: %i\n", temp);
	ibask(atoi(config.gpib_board), IbaSAD, &temp);
	fprintf(logFile, "  IbaSAD: %i\n", temp);
	ibask(atoi(config.gpib_board), IbaTMO, &temp);
	fprintf(logFile, "  IbaTMO: %i\n", temp);
	ibask(atoi(config.gpib_board), IbaEOT, &temp);
	fprintf(logFile, "  IbaEOT: %i\n", temp);
	ibask(atoi(config.gpib_board), IbaPPC, &temp);
	fprintf(logFile, "  IbaPPC: %i\n", temp);
	ibask(atoi(config.gpib_board), IbaAUTOPOLL, &temp);
	fprintf(logFile, "  IbaAUTOPOLL: %i\n", temp);
	ibask(atoi(config.gpib_board), IbaSC, &temp);
	fprintf(logFile, "  IbaSC: %i\n", temp);
	ibask(atoi(config.gpib_board), IbaSRE, &temp);
	fprintf(logFile, "  IbaSRE: %i\n", temp);
	ibask(atoi(config.gpib_board), IbaPP2, &temp);
	fprintf(logFile, "  IbaPP2: %i\n", temp);
	ibask(atoi(config.gpib_board), IbaTIMING, &temp);
	fprintf(logFile, "  IbaTIMING: %i\n", temp);
	ibask(atoi(config.gpib_board), IbaReadAdjust, &temp);
	fprintf(logFile, "  IbaReadAdjust: %i\n", temp);
	ibask(atoi(config.gpib_board), IbaWriteAdjust, &temp);
	fprintf(logFile, "  IbaWriteAdjust: %i\n", temp);
	ibask(atoi(config.gpib_board), IbaSendLLO, &temp);
	fprintf(logFile, "  IbaSendLLO: %i\n", temp);
	ibask(atoi(config.gpib_board), IbaPPollTime, &temp);
	fprintf(logFile, "  IbaPPollTime: %i\n", temp);
	ibask(atoi(config.gpib_board), IbaEndBitIsNormal, &temp);
	fprintf(logFile, "  IbaEndBitIsNormal: %i\n", temp);
}

// PPMS record measurement
bool PPMSWriteMeasurement() {
	char response[4096];		// response buffer
	long responseSize=4096;	// sizeof response buffer
	char rbQuery[13] = { 0 };
	int rbQueryNum = 0;
	int i, j = 0;
	float avgTimeStamp = 0.0f;
	float avgTemp = 0.0f, avgField = 0.0f;
	float* TimeStamps = NULL;
	float* Temps = NULL;
	float* Fields = NULL;
	float* BridgeRorV[3] = { NULL };
	float avgBridgeRorV[3] = { 0.0f };
```

```
	float stdDevBridgeRorV[3] = { 0.0f };
	float* BridgeI[3] = { NULL };
	float avgBridgeI[3] = { 0.0f };


// use ppms rb is true if gpib.use_ppms_resistivity_bridge == TRUE, otherwise it
is false
	bool use_ppms_rb = false;
	int rb_num_readings = 0;
	int num_bridge_read = 0;
	if (strncmp(strupr(config.use_ppms), "TRUE", 4) == 0){
	  if (strncmp(strupr(config.use_ppms_resistivity_bridge), "TRUE", 4) == 0){
		use_ppms_rb = true;
		rb_num_readings = atoi(config.rb_num_readings);
		fprintf(logFile, "PPMSWriteMeasurement: rb_num_readings %i \n",
			rb_num_readings);
		num_bridge_read = atoi(config.rb_num_bridge_read);
		TimeStamps = (float*)malloc(rb_num_readings*sizeof(float));
		Temps = (float*)malloc(rb_num_readings*sizeof(float));
		Fields = (float*)malloc(rb_num_readings*sizeof(float));
		for (i = 0; i < rb_num_readings; i++){
			TimeStamps[i] = 0.0f;
			Temps[i] = 0.0f;
			Fields[i] = 0.0f;
			}
			for (i = 0; i < 3; i++){
				BridgeRorV[i] =
	(float*)malloc(rb_num_readings*sizeof(float));
			BridgeI[i] = (float*)malloc(rb_num_readings*sizeof(float));
				for (j = 0; j < rb_num_readings; j++){
					BridgeRorV[i][j] = 0.0f;
					BridgeI[i][j] = 0.0f;
				}
			}
			rbQueryNum = 6; // Temp + Field
			if (strncmp(strupr(config.rb_read_bridge_1), "TRUE", 4) == 0)
rbQueryNum += 48;
			if (strncmp(strupr(config.rb_read_bridge_2), "TRUE", 4) == 0)
rbQueryNum += 192;
			if (strncmp(strupr(config.rb_read_bridge_3), "TRUE", 4) == 0)
rbQueryNum += 768;
			sprintf(rbQuery, "GETDAT? %i", rbQueryNum);
		}

		if (!use_ppms_rb){ // USE_PPMS_RESISTIVITY_BRIDGE is false
		// Read PPMS Temperature
		if (PPMSSendAndReceive("GETDAT? 6", response, responseSize) <= 16) {
			fprintf(logFile, "PPMSWriteMeasurement: Error querying for
				system temperature and field!\n");
				return false;
			}
		if (sscanf(response, "%*i,%f,%f,%f", &avgTimeStamp, &avgTemp,
			&avgField) < 3) {
			fprintf(logFile, "PPMSWriteMeasurement: Warning, was not able
				to read temperature and field at timestamp %f!\n",
				avgTimeStamp);
			}
		}else{ // USE_PPMS_RESISTIVITY_BRIDGE is true
```

```
                        for (i = 0; i < rb_num_readings; i++) {
                        // Read Timestamp, Temperature, Field, and Bridge(s)
                                if (PPMSSendAndReceive(rbQuery, response, responseSize)
                                        < 1) {
                                        fprintf(logFile, "PPMSWriteMeasurement: Error
                                                running bridge query!\n");
                                        return false;
                                }
                                if (sscanf(response, "%*i,%f,%f,%f,%f,%f,%f,%f,%f,%f",
&(TimeStamps[i]), &(Temps[i]), &(Fields[i]), &(BridgeRorV[0][i]), &(BridgeI[0][i]),
&(BridgeRorV[1][i]), &(BridgeI[1][i]), &(BridgeRorV[2][i]), &(BridgeI[2][i]) ) <
3+2*num_bridge_read){
                                        fprintf(logFile, "PPMSWriteMeasurement: Warning,
                                                was not able to read all values for b
                                                ridge(s) at timestamp %f!\n",
                                                TimeStamps[i]);
                                        }
                                        Sleep(atoi(config.rb_wait_between_readings));
                        }
                        // Compute average of all values
                        for (i = 0; i < rb_num_readings; i++) {
                                avgTimeStamp += TimeStamps[i];
                                avgTemp += Temps[i];
                                avgField += Fields[i];
                                for (int j = 0; j < num_bridge_read; j++) {
                                        avgBridgeRorV[j] += BridgeRorV[j][i];
                                        avgBridgeI[j] += BridgeI[j][i];
                                }
                        }
                        avgTimeStamp /= (float)rb_num_readings;
                        avgTemp /= (float)rb_num_readings;
                        avgField /= (float)rb_num_readings;
                        for (j = 0; j < num_bridge_read; j++) {
                                avgBridgeRorV[j] /= (float)rb_num_readings;
                                avgBridgeI[j] /= (float)rb_num_readings;
                        }

                        // And then standard deviations of the bridge R/V values
                        for (i = 0; i < rb_num_readings; i++){
                                for ( j = 0; j < num_bridge_read; j++){
                                        stdDevBridgeRorV[j] += ((BridgeRorV[j][i] -
                                        avgBridgeRorV[j])*(BridgeRorV[j][i] -
                                        avgBridgeRorV[j]));
                                }
                        }
                for (j = 0; j < num_bridge_read; j++) {
                                stdDevBridgeRorV[j] /= (float)(rb_num_readings);
                                stdDevBridgeRorV[j] = sqrtf(stdDevBridgeRorV[j]);
                                }
                                for (j = 0; j < num_bridge_read; j++) {
                                        fprintf(logFile, "PPMSWriteMeasurement: Values in
                                                stdDevBridgeRorV entry %i: %f \n", j,
                                                stdDevBridgeRorV[j]);
                                }
        // And finally re-index to the actual bridges read for file output
                                switch (num_bridge_read) {
                                case 1:
```

```
                        if (strncmp(strupr(config.rb_read_bridge_3), "TRUE", 4)
                                == 0) {
                                avgBridgeRorV[2] = avgBridgeRorV[0];
                                stdDevBridgeRorV[2] = stdDevBridgeRorV[0];
                                avgBridgeI[2] = avgBridgeI[0];
                                avgBridgeRorV[0] = 0.0f; stdDevBridgeRorV[0]
                                =0.0f; avgBridgeI[0] = 0.0f;
                        } else if (strncmp(strupr(config.rb_read_bridge_2),
                                "TRUE", 4) == 0) {
                                avgBridgeRorV[1] = avgBridgeRorV[0];
                                stdDevBridgeRorV[1] = stdDevBridgeRorV[0];
                                avgBridgeI[1] = avgBridgeI[0];
                                avgBridgeRorV[0] = 0.0f; stdDevBridgeRorV[0]
                                =0.0f; avgBridgeI[0] = 0.0f;
                        }
// otherwise only bridge read is bridge 1, values already in right place
                        break;
                case 2:
                        if (strncmp(strupr(config.rb_read_bridge_2), "TRUE",
                                4) != 0 ||
                               (strncmp(strupr(config.rb_read_bridge_1), "TRUE",
                                4) != 0)) {
//bridge 1 or 2 is odd one out, so values in 2 are for bridge 3
                                avgBridgeRorV[2] = avgBridgeRorV[1];
                                stdDevBridgeRorV[2] = stdDevBridgeRorV[1];
                                avgBridgeI[2] = avgBridgeI[1];
                                avgBridgeRorV[1] = 0.0f; stdDevBridgeRorV[1]
                                      =0.0f; avgBridgeI[1] = 0.0f;
                        }
                        if (strncmp(strupr(config.rb_read_bridge_1), "TRUE",
                                4) != 0) { // bridge 1 is odd one out
                                avgBridgeRorV[1] = avgBridgeRorV[0];
                                stdDevBridgeRorV[1] = stdDevBridgeRorV[0];
                                avgBridgeI[1] = avgBridgeI[0];
                                avgBridgeRorV[0] = 0.0f; stdDevBridgeRorV[0]
                               =0.0f; avgBridgeI[0] = 0.0f;
                        }
// otherwise only bridges read are 1 and 2, values in right place
                        break;
                case 0:
                case 3:
                default:
                // Do nothing: none or all read
                        num_bridge_read = num_bridge_read;
                }
        }

        for (j = 0; j < num_bridge_read; j++) {
                fprintf(logFile, "PPMSWriteMeasurement: Values in
                stdDevBridgeRorV entry %i: %f \n", j,
                stdDevBridgeRorV[j]);
        }

        // Output results
        fprintf(dataFile, "%.1f,%.3f,%.3f", avgTimeStamp, avgTemp,
                avgField);
}
```

```
	if (strncmp(strupr(config.use_cornerstone260), "TRUE", 4) == 0)
	fprintf(dataFile, ",%i,%i,%i,%i", CS_curwaveln, CS_curfilter, CS_curgrating,
		CS_curshutter);
	if (strncmp(strupr(config.use_NOVA), "TRUE", 4) == 0) fprintf(dataFile,
		",%e", NOVA_last_value);
	if (strncmp(strupr(config.use_k2400), "TRUE", 4) == 0) fprintf(dataFile,
		",%e", k2400_last_value);
	if (strncmp(strupr(config.use_ppms_resistivity_bridge), "TRUE", 4) == 0)
	fprintf(dataFile, ",%.4e,%.4e,%.4e,%.4e,%.4e,%.4e,%.4e,%.4e,%.4e",
		avgBridgeRorV[0], stdDevBridgeRorV[0], avgBridgeI[0],
		avgBridgeRorV[1], stdDevBridgeRorV[1], avgBridgeI[1],
		avgBridgeRorV[2], stdDevBridgeRorV[2], avgBridgeI[2]);
	if (strncmp(strupr(config.use_SRS830), "TRUE", 4) == 0) fprintf(dataFile,
		",%e,%e", SRS830_last_chan1, SRS830_last_phase);
	fprintf(dataFile, "\n");
	// Free memory if needed
	if (use_ppms_rb) {
		for (i = 0; i < 3; i++){
			free(BridgeRorV[i]);
			free(BridgeI[i]);
		}
		free(TimeStamps);
		free(Temps);
		free(Fields);
	}
	return true;

}


void PPMSWriteDataHeader() {
	if (strncmp(strupr(config.use_ppms), "TRUE", 4) == 0){
		fprintf(dataFile, "Timestamp, Temp (K), Field (Oe)");
	}
	if (strncmp(strupr(config.use_cornerstone260), "TRUE", 4) == 0){
		fprintf(dataFile, ", Wavelength (nm), Light Filter, Light Grating,
Shutter");
	}
	if (strncmp(strupr(config.use_NOVA), "TRUE", 4) == 0){
		fprintf(dataFile, ", NOVA Power (watts)");
	}
	if (strncmp(strupr(config.use_k2400), "TRUE", 4) == 0){
		fprintf(dataFile, ", k2400 Voltage (V)");
	}
	if (strncmp(strupr(config.use_ppms_resistivity_bridge), "TRUE", 4) == 0)
	{
		fprintf(dataFile, ", Bridge 1 (Ohm or mV), Bridge 1 Std. Dev.,
Bridge 1 Current (uA), Bridge 2 (Ohm or mV), Bridge 2 Std. Dev., Bridge 2 Current
(uA), Bridge 3 (Ohm or mV), Bridge 3 Std. Dev., Bridge 3 Current (uA)");
	}
	if (strncmp(strupr(config.use_SRS830), "TRUE", 4) == 0)
	{
		fprintf(dataFile, ", SRS 830 Chan1 (x), SRS 830 Chan2 (y)");
	}
	fprintf(dataFile, "\n");
		return;
}
```

"sequence.cpp"

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "MCControlv9.h"

#define SEQUENCE_MEM_BLOCK 4096

long *seqlist;
long *seqlist_original;

bool ReadSeqFile(){
       errno_t err;
       FILE* sfile;
       long k, i = 0;
       if (err = fopen_s(&sfile, "sequence.seq", "rt") != 0){
              fprintf(logFile, "sequence: failed to open sequence.seq \n");
              return false;
       }
       seqlist = (long *)malloc(SEQUENCE_MEM_BLOCK*sizeof(long));
       seqlist_original = (long *)malloc(SEQUENCE_MEM_BLOCK*sizeof(long));
       fprintf(logFile, "ReadSeqFile: opened sequence and allocated memory to
seqlist.seq \n");
       while (!feof(sfile) && !ferror(sfile)) {
              fscanf (sfile, "%ld", &i);
              seqlist[k] = i;
              seqlist_original[k] = i;
              k++;
              if (k%SEQUENCE_MEM_BLOCK == 0) {
                     realloc(seqlist,
SEQUENCE_MEM_BLOCK*sizeof(long)*(k/SEQUENCE_MEM_BLOCK + 1));
                     realloc(seqlist_original,
SEQUENCE_MEM_BLOCK*sizeof(long)*(k/SEQUENCE_MEM_BLOCK + 1));
                     fprintf(logFile, "ReadSeqFile: allocated another block \n");
                     if (seqlist == NULL) {
                            fprintf(logFile, "ReadSeqFile: Could not expand array
                                   to size %i",
SEQUENCE_MEM_BLOCK*sizeof(long)*(k/SEQUENCE_MEM_BLOCK + 1));
                            exit(2);
                     }
              }
    }
       seqlist[k] = NULL;
       seqlist_original[k] = NULL;
       int j = 0;
       fprintf(logFile, "ReadSeqFile: This is seqlist ");
       while (seqlist[j] != NULL){
              fprintf(logFile, " %i", seqlist[j]);
              j++;
       }
       fprintf(logFile, "\n");
       if (fclose(sfile)<0) fprintf(logFile, "ReadSeqFile: failed to close
              configfile \n");
       return true;
}
```

"SRS_830.cpp"

```
#include "MCControlv9.h"

////////////////////////////////////////////////////////////////////////////
// Cornerstone 260 global variables
////////////////////////////////////////////////////////////////////////////

// Cornerstone 260
bool SRS830_initialized = false;
int SRS830_addr = -1;
int SRS830_dev = -1;
float SRS830_last_chan1 = 0.0f;
float SRS830_last_phase = 0.0f;

////////////////////////////////////////////////////////////////////////////

//sends Auto Gain command to SRS830.  Return false if write fails, otherwise
return true
bool SRS830_auto_gain(){
	char cmdstr[256] = { 0 };
	sprintf(cmdstr, "AGAN \n");
	if (!ibwrite(SRS830_dev, cmdstr, atoi(config.SRS830_write_sleep))) {
		fprintf(logFile, "SRS830_auto_gain: Could not send AGAN
			command!\n");
		return false;
	}
	fprintf(logFile, "SRS830_auto_gain: Sent AGAN command!\n");
	return true;
}


bool SRS830_clear_failed_measurement(){
	char vread[62] = { NULL };
	if (!ibread(SRS830_dev, vread, 62)) {
		fprintf(logFile, "SRS830_clear_failed_measurement: Could not read
current display values!\n");
		SRS830_clear_failed_measurement();
		return false;
	}
	return true;
}

bool SRS830_meas() {
	if (strncmp(strupr(config.SRS830_auto_gain), "TRUE", 4) == 0){
		SRS830_auto_gain();
	}
	// command string to pass as argument to ibwrt
	char cmdstr1[256] = { 0 };
	char vread1[62] = { NULL };
	sprintf(cmdstr1, "SNAP?10,11 \n");
	if (!ibwrite(SRS830_dev,cmdstr1,atoi(config.SRS830_write_sleep))) {
		fprintf(logFile, "SRS830_meas: Could not send SNAP? command!\n");
		return false;
	}
	if (!ibread(SRS830_dev, vread1, 62)) {
		fprintf(logFile, "SRS830_meas: Could not read current display
			values!\n");
		SRS830_clear_failed_measurement();
```

```
		SRS830_meas();
		return false;
	}
	fprintf(logFile, "SRS830_meas: this is vread1 %s!\n", vread1);
	sscanf(vread1, "%f,%f", &SRS830_last_chan1, &SRS830_last_phase);
	fprintf(logFile, "SRS830_meas: this is SRS830_last_chan1: %f and
		SRS830_last_chan2: %f! \n", SRS830_last_chan1, SRS830_last_phase);
	return true;
}


bool init_SRS830() {
	char cmdstr[256] = { 0 };
	char cmdstr2[256] = { 0 };
	char cmdstr3[256] = { 0 };
	char vread[256] = { 0 };
	if (SRS830_initialized == true) return true;
	fprintf(logFile, "SRS830_init: Beginning Initialization.\n");
	SRS830_addr = atoi(config.SRS830_gpib);
	if ((SRS830_addr < 1) || (SRS830_addr > 30)) {
		fprintf(logFile, "SRS830_init: No or Invalid GPIB address %i
		specified!\n", CS_addr);
		return false;
	}
	fprintf(logFile, "SRS830_init: GPIBaddress  = %i.\n", SRS830_addr);
	SRS830_dev = ibdevice(SRS830_addr, 0 /* No Secondary Address */, 13 /* 30
		sec timeout */, 0 /* Enable END */, GPIB_READ_TERM_CHAR /* End of
		string code */);
	if (SRS830_dev < 1) {
		fprintf(logFile, "SRS830_init: Could not open communications with
			device!\n");
		return false;
	}

	sprintf(cmdstr, "OUTX 1\n");
	if (!ibwrite(SRS830_dev, cmdstr, atoi(config.SRS830_write_sleep))) {
		fprintf(logFile, "SRS830_init: Could not send OUTx 1 command!\n");
		return false;
	}
	sprintf(cmdstr2, "SEND 0\n");
	if (!ibwrite(SRS830_dev, cmdstr2, atoi(config.SRS830_write_sleep))) {
		fprintf(logFile, "SRS830_init: Could not send SEND 0 command!\n");
		return false;
	}
	sprintf(cmdstr3, "SEND?\n");
	if (!ibwrite(SRS830_dev, cmdstr3, atoi(config.SRS830_write_sleep))) {
		fprintf(logFile, "SRS830_init: Could not send SEND 0 command!\n");
		return false;
	}
	if (!ibread(SRS830_dev, vread, 256)) {
		fprintf(logFile, "SRS830_init: Could not read send status!\n");
		return false;
	}
	fprintf(logFile, "SRS830_init: send status is %s!\n", vread);

	// At this point we are initialized and can accept commands through CS_do
	SRS830_initialized = true;
	fprintf(logFile, "SRS830_init: Ready to accept commands.\n");
```

```
        return true;
}
```

"stdadx.cpp"

```
#include "stdafx.h"
```

Chapter 4: Thermally-activated Recombination in one Component of $(CH_3NH_3)PbI_3$ / TiO_2 Observed by Photocurrent Spectroscopy

This work was co-written with the following authors and is published under the following citation:

Patrick Cottingham, David C. Wallace, Ke Hu, Gerald Meyer, and Tyrel M. McQueen *Chem. Commun.* **2015**, *51*, 7309-7312.

4.1 Introduction

Organometallic halide perovskites have been the subject of intense research in recent years. Interest in these materials stems in part from their ability to function as efficient light absorbers and electron- and hole-transport materials in photovoltaic (PV) devices. These materials are especially attractive for PV applications because of their earth-abundant components and the wide variety of low- energy synthetic techniques[1-4] by which they can be produced. Early studies of PV devices containing organometallic halide perovskites used the perovskite $(CH_3NH_3)PbI_3$ deposited into a mesoporous TiO_2 matrix[5-7] (the crystal structure of $(CH_3NH_3)PbI_3$ is given in Fig. 4.1). Later studies demonstrated that efficient devices could be constructed using $(CH_3NH_3)PbI_3$ and larger-bandgap oxides such as Al_2O_3 and ZrO_2 in which the energy of the conduction band minimum is too high for electron injection from $(CH_3NH_3)PbI_3$ to occur.[8-11] Perovskite PV cells not containing any oxide matrix ('planar' devices) have also produced high efficiencies.[2,9,12,13]

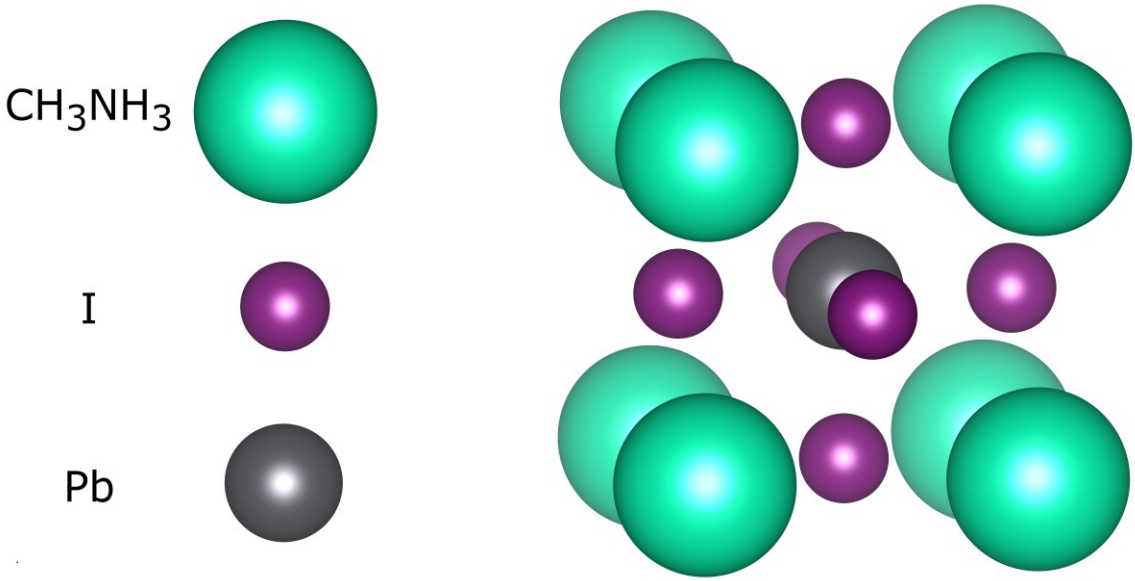


Fig. 4.1 The crystal structure of $(CH_3NH_3)PbI_3$. CH_3NH_3 is depicted as a sphere because the orientation of the $CH_3NH_3^+$ polar cation during device operation is unknown and may vary with material preparation, applied field, and irradiation.

It has been suggested that in addition to providing a driving force for charge separation at the $(CH_3NH_3)PbI_3$ / TiO_2 interface, TiO_2 enhances the performance of devices by acting as an electron-transport material.[14] However, devices with $(CH_3NH_3)PbI_3$ deposited in highly-insulating Al_2O_3, where the conduction band minimum of the oxide is well above that of the perovskite, achieve similar or better performance than devices on TiO_2.[8] Given the performance of Al_2O_3-containing devices, the role of the oxide matrix in charge separation remains unclear.

A recent study of the structure of $(CH_3NH_3)PbI_3$ in a TiO_2 matrix shows that the matrix has a profound effect on the structure of the perovskite.[14] PDF (pair distribution function) analysis of x-ray total scattering data suggests that when $(CH_3NH_3)PbI_3$ is deposited into mesoporous TiO_2 it forms two components with the same crystal structure:

a relatively disordered component consisting of ~1.4 nm diameter nanocrystals which are confined by pores of the matrix and one with much longer-range coherence. More recent work suggests that the catalytic action of water may be essential for the formation of the perovskite phase.[15] In order to explore the role of the oxide and disorder in charge separation, a series of temperature-dependent photocurrent measurements were collected on 'incomplete' solar cells consisting of electrodes and $(CH_3NH_3)PbI_3$ with or without a mesoporous oxide layer as well as on a 'complete' cell which contained the above components and a dense TiO_2 blocking layer.

4.2 Experimental

4.2.1 Fabrication of Incomplete Cells

Methylammonium iodide, $(CH_3NH_3)I$, was synthesized in the following way. 33 wt% methylamine ethanol solution (Aldrich, $D_{25°C}$ = 0.756 g/mL, 28.3 mL, 0.228 mol) and 57 wt% hydroiodic acid in water (Aldrich, $D_{25°C}$ = 1.701 g/mL, 25.0 mL, 0.189 mol) containing hydrophosphous acid as a stabilizer were mixed and stirred at 0 °C to form a clear solution. The solution was dried in a rotary evaporator to form a white precipitate. The solid was washed three times with diethyl ether with drying after each step for a final yield of 89%.

Mesoporous TiO_2 and ZrO_2 thin films on FTO substrates (without a dense/blocking layer) were prepared according to a published procedure.[16] Briefly, 10 mL titanium(IV) isopropoxide (Aldrich, ≥ 97%) was added dropwise to 60 mL di water with 0.42 mL HNO_3 (70% V/V). The solution was heated to ~95 °C and the volume was reduced to ~20 mL (~160 g/L in TiO_2 concentration). The solution was put in an acid

digestion bomb and autoclaved at 200 °C for 12 hours. The resulting product was opaque white in appearance and 0.9 g carbowax (Polyethyleneglycol Bisphenol A Epichlorohydrin Copolymer 15,000 - 20,000 Da, Aldrich) was added. The TiO_2 paste was "doctor-bladed" onto a pre-cleaned fluorine doped tin oxide (FTO, 15 Ω/ sq., Hartford Glass) substrate using Scotch Tape™ as the spacer, dried, and sintered in a tube furnace at 450 °C under constant oxygen flow for 30 minutes. Mesoporous ZrO_2 thin films were made in a similar way except that Zr(IV) propoxide (70% in 1-propanol) was used as the precursor.

40.3% in weight perovskite solution was prepared by adding 0.979 g CH_3NH_3I and 2.864 g PbI_2 (Aldrich, 99%) in 5 mL γ-butyrolactone (Aldrich, ≥ 99%) and stirred under argon at 60 °C overnight. The solution was filtrated through a 0.22 μm PVDF syringe filter (MILLEX®GV). The resulting coating solution was spread on the substrates, allowed to wait for 1 minute, and spin coated onto an FTO substrate (Fig. 4.2a) or mesoporous TiO_2 and ZrO_2 on FTO substrates (Fig. 4.2b) for 45 seconds at speed of 4000 rpm. The coated slides were annealed at 100 °C for 30 minutes and the color turned from light yellow to black.

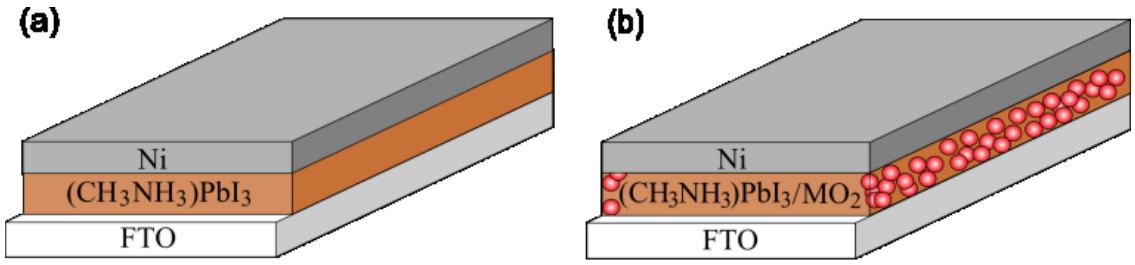


Fig. 4.2 Schematic illustration of two cells containing $(CH_3NH_3)PbI_3$. All cells contain an absorber layer compressed between one FTO electrode and one nickel electrode. The cell in (a) contains a planar $(CH_3NH_3)PbI_3$ absorber layer. The cell in (b) contains $(CH_3NH_3)PbI_3$ embedded in a mesoporous MO_2 matrix, where M is Ti or Zr.

4.2.2 X-Ray Diffraction of $CH_3NH_3PbI_3$ in Mesoporous TiO_2

In order to verify that the PbI_2 used to synthesize $(CH_3NH_3)PbI_3$ was fully reacted and $(CH_3NH_3)PbI_3$ was not decomposed, x-ray diffraction was performed on a sample of $(CH_3NH_3)PbI_3$ / mesoporous TiO_2. Data were collected using a Bruker D8 diffractometer equipped with Cu Kα radiation (λ_1 = 1.540 56 Å, λ_2 = 1.544 39 Å) and a LynxEye detector. This data is shown in Fig. 4.3, along with data collected on powdered PbI_2. Near many values of 2θ where strong reflections occur in the latter pattern, there are no peaks in the former pattern. This absence indicates that no significant fraction of PbI_2 exists in the $(CH_3NH_3)PbI_3$ sample.

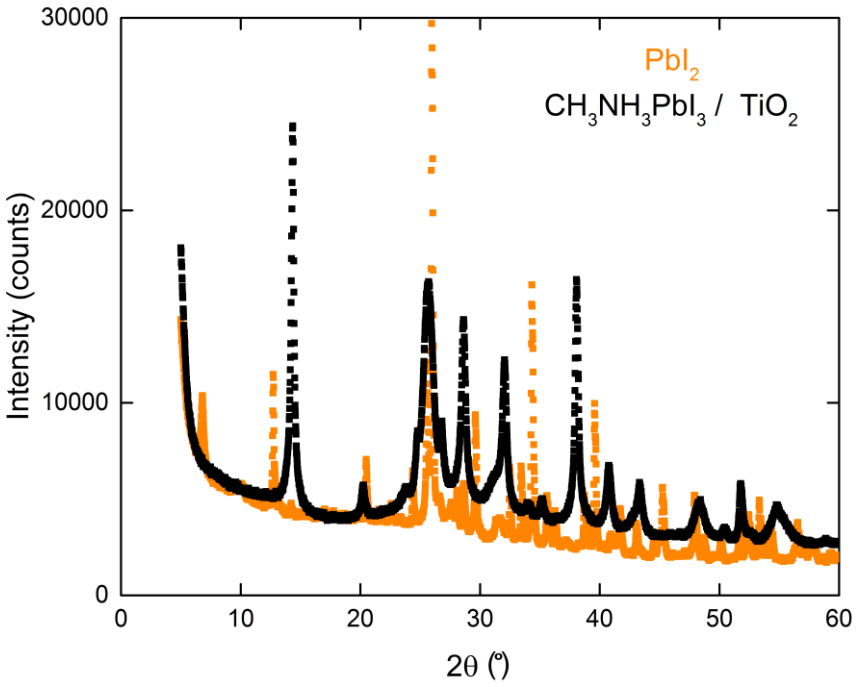


Fig. 4.3 X-ray diffraction patterns of powdered PbI_2 and a sample of $(CH_3NH_3)PbI_3$ in mesoporous TiO_2

4.2.3 Fabrication of Complete Cells

A dense, compact TiO_2 layer was deposited on FTO glass substrates (Hartford Glass, Inc.) in the following way: A 0.15 M solution of titanium(IV) isopropoxide (Aldrich, ≥ 97%) in 2-propanol was spin-coated onto the substrate at 2000 RPM, followed by heating of the substrate at 110 °C for 10 minutes. The previous step was repeated twice with a 0.3 M solution followed by heating at 450 °C for 30 minutes.

$(CH_3NH_3)PbI_3$ was deposited onto the FTO/blocking layer substrates according a variation on the method of Gratzel et al.[3] Briefly, a 1 M suspension of PbI_2 in dimethylformamide (DMF) that had been heated to 70 °C was spin-coated at 3000 RPM onto the FTO and blocking layer substrate that had also been preheated to 70 °C. This

sample was then annealed for 30 minutes on a hotplate at 70 °C. The sample was then swirled in a 100 mg/ml solution of CH_3NH_3I in 2-propanol for 20 seconds. Drying the sample at 75 °C on a hotplate caused an immediate color change to homogenous dark brown. A solution of 10 mg/ml CH_3NH_3I and 1% in weight PEDOT:PSS (Aldrich, from dry pellets) in 2-propanol was spin-coated onto the sample at 2000 RPM in order to create a hole-transport layer. The sample was compressed against another piece of FTO glass and sealed with glue in order to create a complete photovoltaic device (Fig. 4.4).

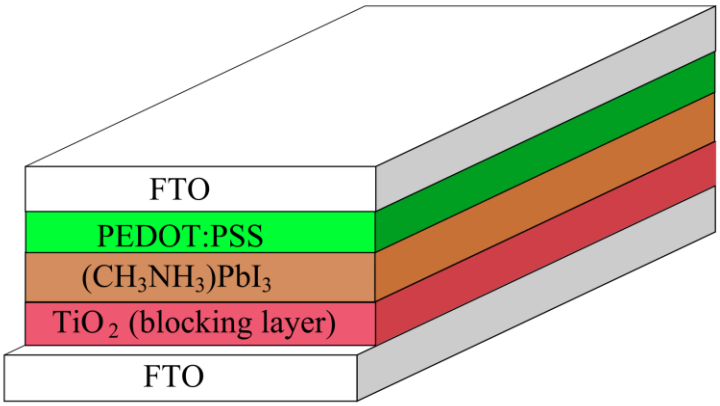


Fig. 4.4 Schematic illustration of a complete cell.

4.2.4 Photocurrent measurements

Photocurrent measurements were conducted by mounting the samples in a bespoke probe which was inserted into a Quantum Design Physical Properties Measurement System (PPMS) in order to achieve precise control of the sample's temperature. This probe is described in significant detail in Chapter 3. An ~10 nm spectral band was selected from a white light source using a monochromator and used to illuminate the sample via an optical fiber. Samples were measured over the wavelength range λ = 350-1100 nm at each temperature, beginning with λ = 350 nm. Scans were

performed at a series of decreasing temperatures, beginning at $T = 300$ K. The light intensity arriving at the sample varied with wavelength and did not exceed 25 nW cm^{-2} at any wavelength. The output of the monochromator was mechanically chopped at 315 Hz and filtered for harmonics. Measurement of the photocurrent was accomplished by measuring the voltage across a resistor (R = 9.8 kΩ or 980 kΩ) connected across the terminals of the device using a Stanford Research Systems SRS830 lock-in amplifier.

4.3 Photocurrent Spectra

4.3.1 Photocurrent Spectra of Incomplete Cells

Fig. 4.5a shows the photocurrent of the device containing planar $(CH_3NH_3)PbI_3$. The photocurrent is approximately constant with wavelength from λ = 400 nm until the photocurrent onset at λ = 800 nm. This spectral shape is commonly associated with the photoconductivity of a direct-bandgap semiconductor. In devices of this kind, containing neither a strong interfacial electric field nor a dedicated electron- or hole-transport layer, charge separation may occur by diffusion.[17] As the temperature of the sample is decreased, the photocurrent decreases at all wavelengths above the photocurrent edge by an approximately constant factor.

Fig. 4.5b shows the photocurrent of a device with mesoporous TiO_2. The photocurrent spectrum for this device contains two distinct responses. The first response is centered at shorter wavelengths and shows a gradual decrease in intensity on scanning to longer wavelengths. The second response has a peak-like shape and is centered at around λ = 760 nm. This kind of distinct upturn near the

band-edge has been previously observed in devices containing $(CH_3NH_3)PbI_3$ and oxide when the data is reported in terms of external quantum efficiency.[2,18,19] The photocurrent of $(CH_3NH_3)PbI_3$ in mesoporous ZrO_2 (Fig. 4.5c) shows two similarly shaped responses.

A plausible explanation for the appearance of two responses in the photocurrent when $(CH_3NH_3)PbI_3$ is deposited on a mesoporous matrix is the formation of two components as observed by Choi et al.[14]. The peak-like shape of the longer-wavelength response is reminiscent of a quantum dot, suggesting that it derives from the short-range ordered component. However, quantum confinement effects blue-shift the absorption of nanocrystals, suggesting that the photocurrent at shorter wavelengths derives from the short-range ordered component. If the shorter-wavelength response does correspond to the short-range ordered component, the Brus equation[20] for the energy of transitions in a semiconductor cluster (Eqn. 4.1)

$$\Delta E(r) = E_{gap} + \frac{h^2}{8r^2}\left(\frac{1}{m_{e^*}} + \frac{1}{m_{h^*}}\right) \quad \textbf{(4.1)}$$

can be used to calculate the effective mass of a free carrier in this component by assuming that the electron and hole effective masses, m_e^* and m_h^*, are equal. $\Delta E(r)$ is the energy of the transition in the cluster, E_{gap} is the bandgap of the bulk semiconductor, r is the radius of a cluster, and h is the Planck constant. The effective mass calculated using this equation is $m^* = 15\ m_0$, which differs substantially from the values of $m_e^* = 0.23\ m_0$ and $m_h^* = 0.29\ m_0$ determined from density functional theory (DFT) calculations[21]. This discrepancy suggests that the difference in the photocurrent edge between the two components derives from

differences in structure, composition, or disorder or from space-charge effects rather than quantum confinement.

As it is not possible to definitively assign the two features in the photocurrent spectrum to either the short-range ordered or the medium-range ordered component of $(CH_3NH_3)PbI_3$, they will be referred to below as component 1 (giving rise to the feature at shorter wavelengths) and component 2 (giving rise to the feature centered at $\lambda = 760$ nm).

Plotting the photocurrent at $\lambda = 500$ nm and $\lambda = 760$ nm as a function of temperature shows profound differences between different devices. For planar $(CH_3NH_3)PbI_3$ (Fig. 4.5d), the responses at both wavelengths vary almost linearly with temperature and have approximately the same positive slope. For $(CH_3NH_3)PbI_3$ in TiO_2, however, the response at $\lambda = 500$ nm *decreases* with increasing temperature while the response at $\lambda = 760$ nm increases with increasing temperature (Fig. 4.5e). For $(CH_3NH_3)PbI_3$ in ZrO_2, the two response at both wavelengths both increase with increasing temperature (Fig. 4.5f).

The short-wavelength photocurrent efficiency decreases with increasing temperature for the TiO_2-containing devices and increases for all other samples; this implies that TiO_2 is responsible for this behaviour.

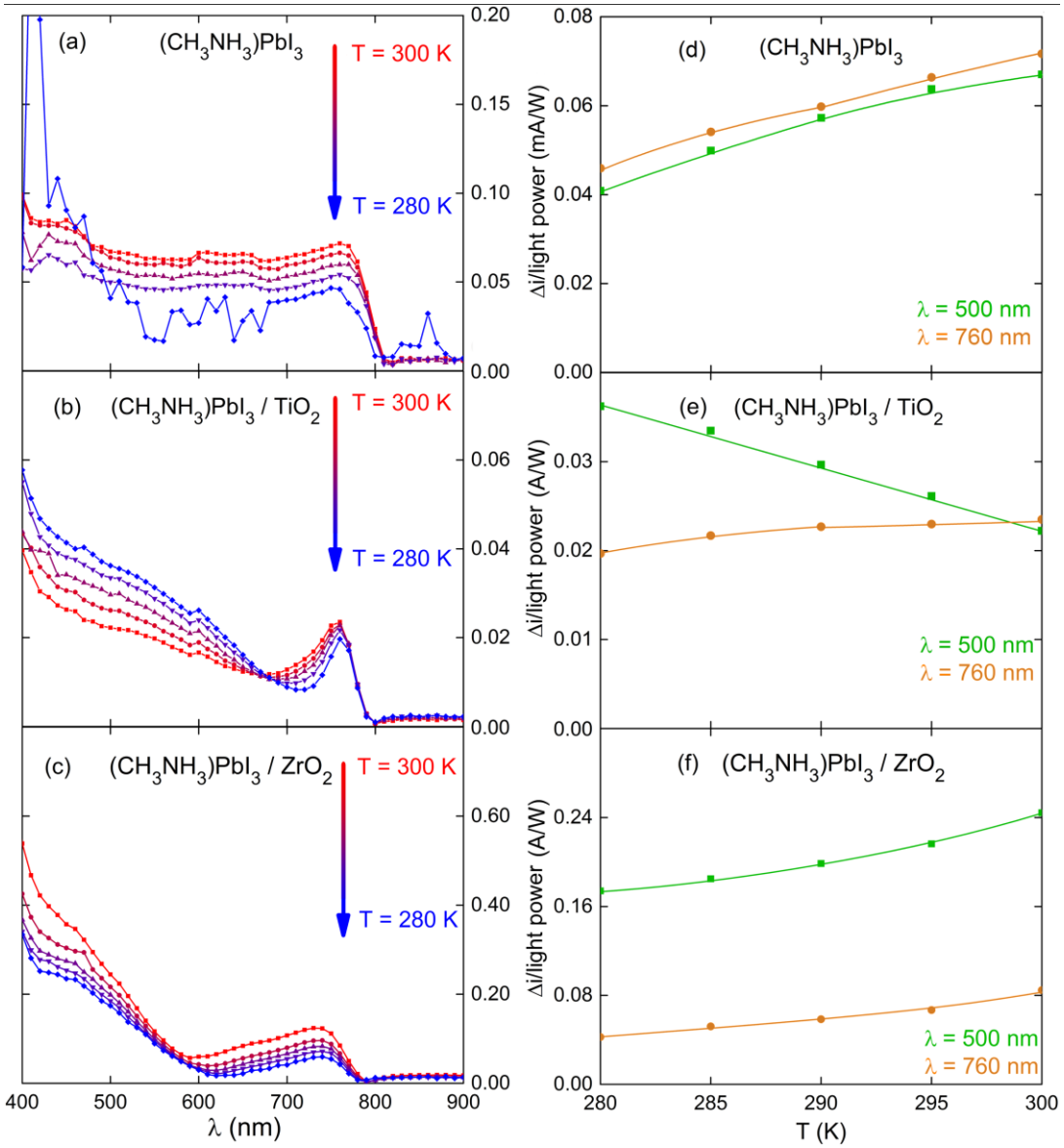


Fig. 4.5 (a), **(b)**, and **(c)** respectively show the photocurrent of devices containing planar $(CH_3NH_3)PbI_3$, $(CH_3NH_3)PbI_3$ in TiO_2, and $(CH_3NH_3)PbI_3$ in ZrO_2 as a function of temperature. **(d)**, **(e)**, and **(f)** show the photocurrent of those devices at $\lambda = 500$ nm, and $\lambda = 760$ nm as a function of temperature. All lines are guides to the eye. Noise at $T = 280$ K is due to failure of the light bulb.

The changes in photocurrent at λ = 500 nm for the TiO_2-containing devices can be described well by an expression for thermally activated recombination containing a constant term which represents the photocurrent in the absence of recombination and an Arrhenius-type term which accounts for recombination (Eqn. 4.2). Here Δi is the measured photocurrent normalized for the incident light intensity, E_a is the energy barrier for recombination, C is a constant which represents the photocurrent in the absence of recombination, and A is a free scaling parameter. The sign of A is opposite to that of Δi and C because the Arrhenius term represents recombination which reduces the overall photocurrent.

$$\Delta i = C + Ae^{-E_a/k_BT} \qquad (\mathbf{4.2})$$

If the value of C = 0.06(1) A/W determined from fitting is subtracted from Δi, the Arrhenius term can be isolated and shown in a typical ln(k) vs. 1/T -type plot (Fig. 4.6a). The energy barrier determined from the fit is E_a = 0.17(5) eV. The results of the fit can be used to construct a model of charge recombination for the three incomplete devices (Fig. 4.6b). For all of the devices, free charge carriers are generated when the absorption of a photon generates an electron-hole pair. For planar $(CH_3NH_3)PbI_3$, component 2 of $(CH_3NH_3)PbI_3$ in TiO_2, and both components of $(CH_3NH_3)PbI_3$ in ZrO_2, recombination proceeds through a pathway in which thermal activation is not the rate-limiting step. In contrast, for component 1 of $(CH_3NH_3)PbI_3$ in TiO_2, recombination occurs with an effective energy barrier of E_a = 0.17(5) eV.

The absence of thermally-activated recombination in component 2 of $(CH_3NH_3)PbI_3$ in TiO_2 suggests that charge collection from this component

proceeds through a pathway not involving TiO_2, perhaps involving percolation through a network of nano-scale domains. It is plausible for such networks to exist given Choi et al.'s observation[14] that ~70% of $(CH_3NH_3)PbI_3$ on TiO_2 consists of the short-range ordered component.

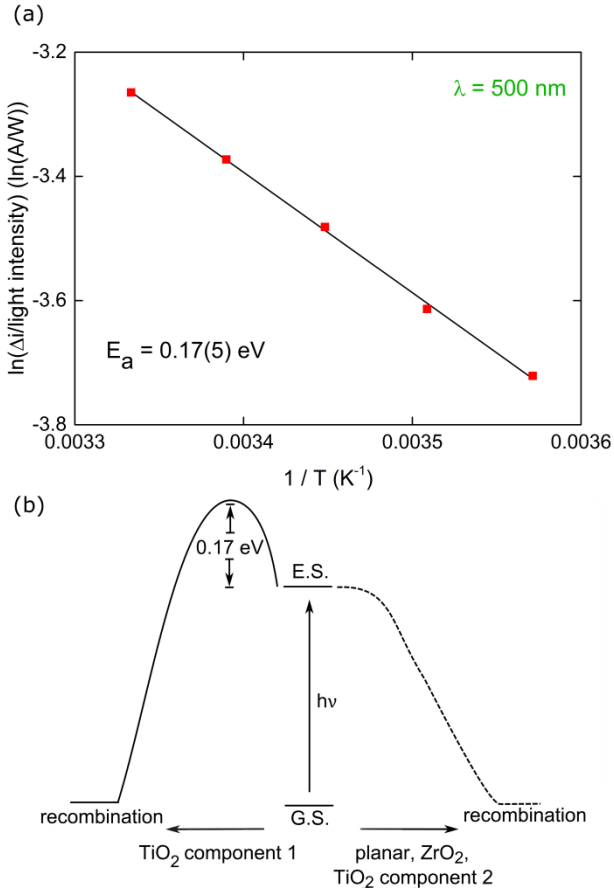


Fig. 4.6 (a) Shows a fit to the change in photocurrent as a function of temperature for the device containing $(CH_3NH_3)PbI_3$ in TiO_2 at $\lambda = 500$ nm. **(b)** Shows a model of charge recombination in component 1 of $(CH_3NH_3)PbI_3$ in TiO_2, compared with planar $(CH_3NH_3)PbI_3$, $(CH_3NH_3)PbI_3$ in ZrO_2, and component 2 of $(CH_3NH_3)PbI_3$ in TiO_2.

The thermally-activated recombination of carriers from component 1 of $(CH_3NH_3)PbI_3$ in TiO_2 likely consists of a combination of back-electron transfer and/or recombination related to transport through TiO_2. Experimentally determined values for the offset between the conduction band minima of $(CH_3NH_3)PbI_3$ and TiO_2 range from 0.07 eV[7] to 0.4 eV.[22] DFT including spin-orbit coupling has been used to calculate an offset of 0.2 eV.[23] With an energy offset in this range electron transfer from TiO_2 to $(CH_3NH_3)PbI_3$ is possible. This is in contrast to ZrO_2 where the conduction band minimum is more than 1 eV above that of TiO_2[24] and electron transfer to the oxide is unlikely. Studies of transport in TiO_2-based dye-sensitized solar cells (DSSCs) have also found activation energies for electron mobility of E_a = 0.15 eV.[25] The thermal activation of recombination may involve transport to recombination sites in TiO_2 or to an interface where recombination or back-electron transfer occurs.

It is unclear from the data above why charge injection to TiO_2 does not occur from component 2. One explanation is that this component is physically separated from TiO_2, likely by component 1. Alternately, it is possible that the conduction band minimum of component 2 is below that of TiO_2 and carriers diffuse away from the $(CH_3NH_3)PbI_3$ / TiO_2 interface before hot-carrier or thermally-activated injection can occur.

4.3.2 Photocurrent Spectrum of Complete Cell

In order to verify that the appearance of two features in the photocurrent spectra of samples containing a mesoporous oxide matrix depends on infiltration of $(CH_3NH_3)PbI_3$ into the matrix, photocurrent measurements were performed on a

device containing a dense, compact TiO_2 layer, but no mesoporous matrix. This spectra was collected at room temperature, with optical chopping at 50 Hz, using the SRS830 in current mode. In Fig. 4.7 this spectrum is compared with the $T = 300$ K spectrum of the mesoporous TiO_2 device. It is clear from this comparison that the presence of compact TiO_2 adjacent to a planar $(CH_3NH_3)PbI_3$ is insufficient to induce the appearance of two spectral features.

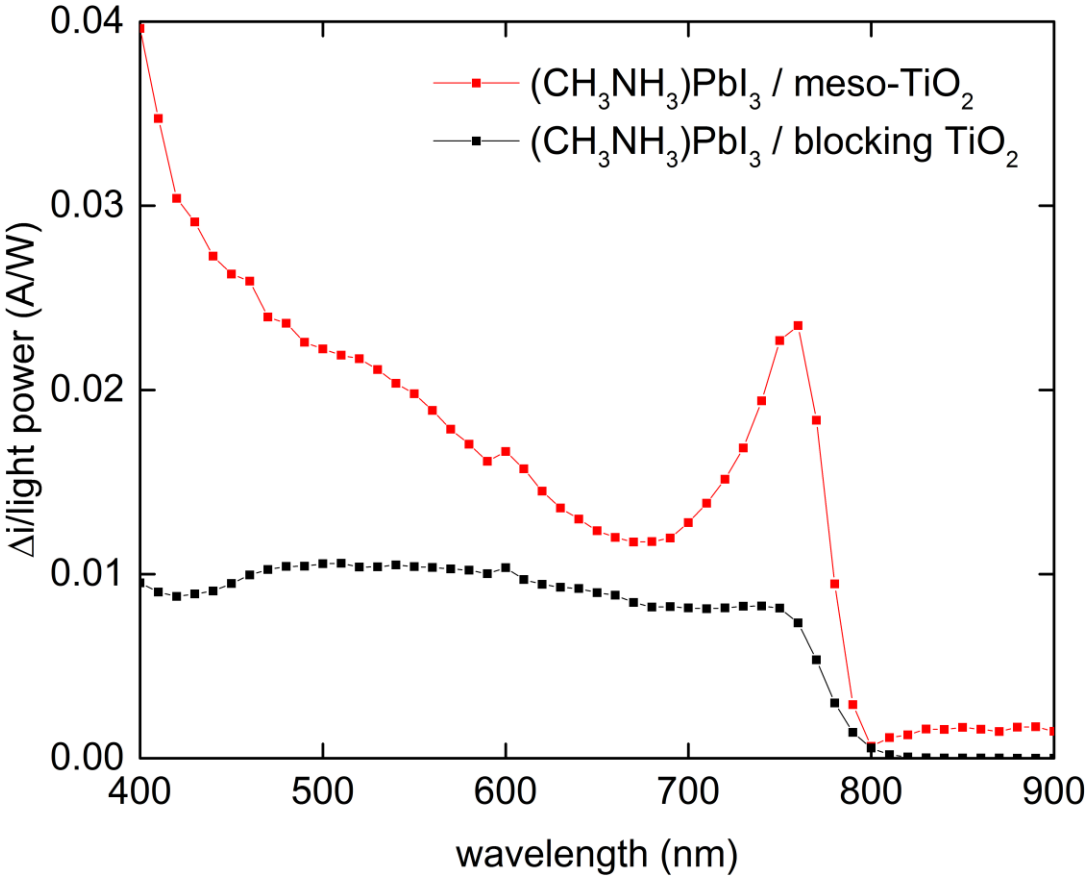


Fig. 4.7 The photocurrent of two devices, one consisting of $(CH_3NH_3)PbI_3$ in a mesoporous TiO_2 matrix (described further in main text) and one containing a dense, compact TiO_2 layer, a planar $(CH_3NH_3)PbI_3$ layer, and a PEDOT:PSS hole transport layer.

4.4 Device Microstructure

One aspect of the photocurrent data which is not fully explained by the model in Fig. 4.6b is the substantially larger photocurrent at all wavelengths of the device containing $(CH_3NH_3)PbI_3$ on ZrO_2. The difference may result from differences in sample thickness. Scanning electron microscopy (SEM) imaging shown in Fig. 4.8 indicates that the TiO_2- containing samples were ~2.5 times thicker than ZrO_2-containing samples. This difference is approximately the same as the difference in photocurrent between the samples at longer wavelengths where injection into TiO_2 is not a factor. Similar decreases in incident photon conversion efficiency (IPCE) have been observed with increasing $(CH_3NH_3)PbI_3$ / TiO_2 layer thickness for complete PV cells and has been attributed to increased recombination.[26]

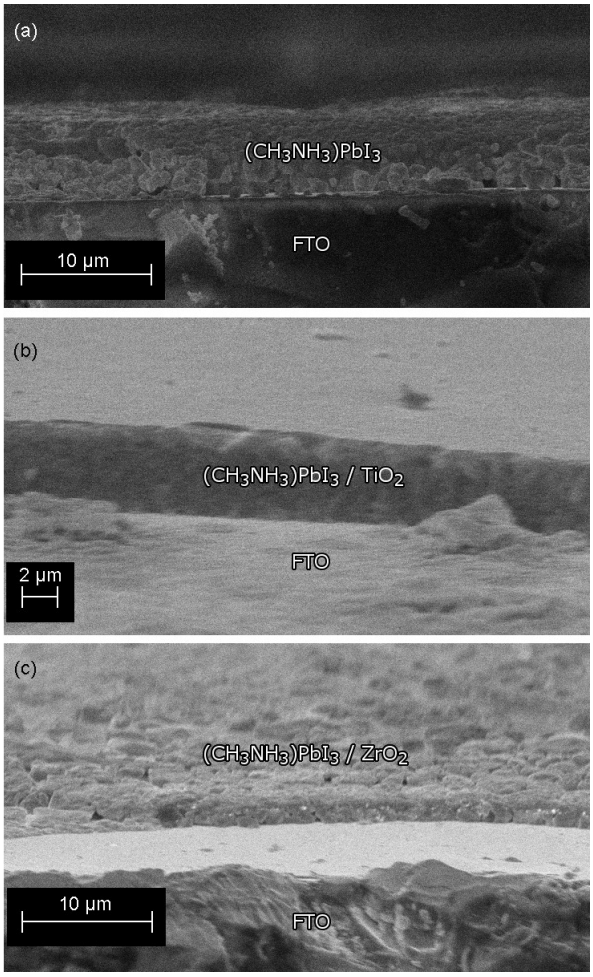


Fig. 4.8 Scanning electron microscope (SEM) images of **(a)** planar $(CH_3NH_3)PbI_3$, **(b)** $(CH_3NH_3)PbI_3$ in TiO_2, and **(c)** $(CH_3NH_3)PbI_3$ in ZrO_2.

4.5 Conclusions

The observation of two distinct responses in the photocurrent spectra of devices containing $(CH_3NH_3)PbI_3$ on an oxide matrix, compared to one response for planar $(CH_3NH_3)PbI_3$, supports previous crystallographic observations of two distinct components in $(CH_3NH_3)PbI_3$ on TiO_2. The inverted temperature dependence of the photocurrent from one component of $(CH_3NH_3)PbI_3$ in TiO_2 indicates that

thermally activated recombination occurs when charge injection and transport occur through TiO_2. The other component of $(CH_3NH_3)PbI_3$ in TiO_2 appears to separate and transport charge through a mechanism not involving TiO_2. Fitting the activation energy of recombination with an Arrhenius term gives an energy barrier for recombination of $E_a = 0.17(5)$ eV.

4.6 References

(1) Kumar, M. H.; Yantara, N.; Dharani, S.; Grätzel, M.; Mhaisalkar, S.; Boix, P. P.; Mathews, N. *Chem. Commun.* **2013**, *49*, 11089-11091.

(2) You, J.; Hong, Z.; Yang, Y.; Chen, Q.; Cai, M.; Song, T.-B.; Chen, C.-C.; Lu, S.; Liu, Y.; Zhu, H.; Yang, Y. *ACS nano* **2014**, *8*, 1674-1680.

(3) J. Burschka, N. Pellet, S.-J. Moon, R. Humphrey-Baker, P. Gao, M. K. Nazeeruddin, and M. Grätzel, *Nature* **2013**, *499*, 316-319.

(4) Docampo, P.; Hanusch, F.; Stranks, S. D.; Döblinger, M.; Feckl, J. M.; Ehrensperger, M.; Minar, N. K.; Johnston, M. B.; Snaith, H. J.; Bein, T.

(5) Kojima, A.; Teshima, K.; Shirai, Y.; Miyasaka, T. *J. Am. Chem. Soc.* **2009**,

(6) Im, J. H.; Lee, C. R.; Lee, J. W.; Park, S. W.; Park, N. G. *Nanoscale*, **2011**, *3*, 4088-4093.

(7) Kim, H.-S.; Lee C.-R.; Im, J.-K.; Lee, K.-B.; Moehl, T.; Marchioro, A.; Moon, S.-J.; Humphry-Baker, R.; Yum, J.-H.; Moser, J. E.; Grätzel, M.; Park, N.-G. *Sci. Rep.* **2012**, *2*, 591.

(8) Lee, M. M.; Teuscher, J.; Miyasaka, T.; Murakami, T. N.; Snaith, H. J. *Science*, **2012**, 338, 643-647.

(9) Ball, J. M.; Lee, M. M.; Hey, A.; Snaith, H. J. *Energy Environ. Sci.*, **2013**, *6*, 1739.

(10) Bi, D.; Moon, S.-J.; Haggman, L.; Boschloo, G.; Yang, L.; Johansson, E. M. J.; Nazeeruddin, M. K.; Grätzel, M.; Hagfeldt, A. *RSC Adv.,* **2013**, *3*, 18762-18766.

(11) Kim, H.-S.; Mora-Sero, I.; Gonzalez-Pedro, H.-S.; Fabregat-Santiago, F.; Juarez-Perez, E. J.; Park, N.-G.; Bisquert, J. *Nat. Commun.*, **2013**, *4*, 2242.

(12) Liu, M.; Johnston, M. B.; Snaith, H. J. *Nature*, **2013**, *501*, 395-398.

(13) Edri, E.; Kirmayer, S.; Henning, A.; Mukhopadhyay, S.; Gartsman, K; Rosenwaks, Y.; Hodes, G.; Cahen, D. *Nano Lett.* **2014**, *14*, 1000-1004.

(14) Choi, J. J.; Yang, X.; Norman, Z. M.; Bilinge, S. J. I.; Owen, J. S. *Nano Lett.* **2014**, *14*, 127-133.

(15) K. K. Bass, R. E. McAnally, S. Zhou, P. Djurovich, M. E. Thompson, and B. Melot, *Chem. Commun.*, **2014**, *50,* 15819.

(16) Argazzi, R.; Bignozzi, C. A.; Heimer, T. A.; Castellano, F. N.; Meyer, G. J. *Inorg. Chem.* **1994**, *33*, 5741-5749

(17) T. Ripolles-Sanchis, A. Guerrero, J. Bisquert, G. Garcia-Belmonte, *J. Phys. Chem. C*, **2012**, *116*, 16933.

(18) Abrusci, A.; Stranks, S.D.; Docampo, P.; Yip, H.-L.; Jen, A. K.-Y.; Snaith, H.J. *Nano. Lett.* **2013**, *13*, 3124-3128.

(19) Zhou, H.; Chen, W.; Li, G.; Luo, S.; Song, T. -B.; Duan, H. -S.; Hong, Z.; You, J.; Liu, Y; Liu, T.; Yang, T. *Science*, **2014**, *345*, 542-546.

(20) Brus, L. *J. Phys. Chem,* **1986**, *90*, 2555-2560.

(21) Giorgi, G.; Fujisawa, J. -I.; Segawa, H.; Yamashita, K. *J. Phys. Chem Lett.*, **2013**, *4*, 4213-4216.

(22) Lindblad, R.; Bi, D.; Park, B.-W.; Oscarsson, J.; Gorgoi, M.; Siegbahn, H.; Odelius, M.; Johansson, E. M. J.; Rensmo, H. *J.Phys.Chem.Lett*. **2014**, *5*, 648-653.

(23) Evan, J.; Pedesseau, L.; Jancu, J.-M.; Katan, C. *J. Phys. Chem. Lett.*, **2013**, *4*, 2999-3005.

(24) Chai, L.; White, R.T.; Greiner, M.T.; Lu, Z. H. *Phys. Rev. B* **2014**, *89*, 035202.

(25) Agrell, H. G.; Boschloo, G.; Hagfeldt, A. *J. Phys. Chem.B.* **2004**, *108*, 12388-12396.

(26) Heo, J. H.; Im, S. H.; Noh, J. H.; Mandal, T. N.; Lim, C. -S.; Chang, J. A.; Lee, Y. H.; Kim, H. -J.; Sarkar, A.; Nazeeruddin, Md. K.; Grätzel, M.; Seok, S. I. *Nat. Photonics*, **2013**, *7*, 486-491.

Curriculum Vitae

Patrick Landon Cottingham

MA, Chemistry	Johns Hopkins University	2012
MSci, Renewable Energy	Loughborough University	2010
BS	Duke University	2003

Employment

Teaching Assistant, Johns Hopkins University	2010-2013
Researcher, Energy Frontier Research Center at the University of North Carolina- Chapel Hill	2009
Laboratory Manager, Johns Hopkins University,	2006-2008
Senior Laboratory Technician, Johns Hopkins University	2004-2006

Research Interests

Photoconductivity and charge separation in inorganic materials and thin-film devices. Novel photovoltaic devices incorporating materials with strongly-correlated electrons. Instrument design for optoelectronic measurements. Discovery of materials featuring superconducting and charge-density wave phases.

Selected Publications

Fuhrman, W. T.; Leiner, J.; Nikolić, P.; Granroth, G. E.; Stone, M. B.; Lumsden, M. D.; DeBeer-Schmidt, L.; Alekseev, P. A.; Mignot, J.-M.; Koohpayeh, S. M.; Cottingham, P.; Phelan, W. A.; Schoop, L.; McQueen, T. M.; Broholm, C. *Phys. Rev. Lett.*, **2015**, *114*, 036401.

Cottingham, P.; Wallace, D. C.; Hu, K.; Meyer, G.; McQueen, T. M. *Chem. Commun.*, **2015**, *51*, 7309-7312.

Phelan, W. A.; Koohpayeh, S. M; Cottingham, P.; Freeland, J. W.; Leiner, J. C.; Broholm, C. L.; McQueen, T. M. *Physical Review X*, **2014**, *4*, 031012.

Cottingham, P.; Miller, D. C.; Sheckelton, J. P.; Neilson, J. R.; Feygenson, M.; Huq, A.; McQueen, T. M. *Journal of Materials Chemistry C*, **2014**, *2*, 3238-3246.

Lectures

Cottingham, P.; Sheckelton, J.; Miller, D.; Neilson, J.; McQueen, T. *APS Meeting Abstracts*; APS National Meeting, Baltimore, MD, March 18-22, 2013; American Physical Society: College Park, MD, USA, 2013; abstract #Y19.009.

Patents

McQueen, T. M.; Cottingham, P.; Sheckelton, J. P.; Arpino, K. (Johns Hopkins University, USA). Simplified devices using novel pn-semiconductor structures. US Patent 8,860,078, October 14, 2014.