Instrument Science Report STIS 2018-03(v1)

Investigating the Binary Offset Effect in the STIS CCD

John H. Debes[1], Sean A. Lockwood[1]

[1]*Space Telescope Science Institute, Baltimore, MD*

23 May 2018

ABSTRACT

Recently, Boone et al., (2018) presented the "Binary Offset Effect" for the SNIFS instrument, which uses a CCD detector. The source of this uncertainty is related to the analog-to-digital readout process, which converts the analog electronic signal of the detector into a digital number as represented by binary bits. The Binary Offset Effect is due to cross-talk between the digital conversion process for a source or driver pixel and pixels read out after the driver. In the course of Boone et al.'s experimentation with this effect they identified a similar effect with the STIS CCD. The STIS team has independently investigated the Binary Offset Effect for a range of bias images currently used for scientific observations, broadly confirming that the effect exists. However, our preliminary investigation suggests that the impact is smaller than reported in Boone et al. (2018) for biases taken with Amplifier=D and GAIN=1, and a lesser effect exists for Amplifier=D and GAIN=4. There is a hint that the effect is time variable for the detector. We broadly assess the potential impact of this effect and make recommendations both for users and future directions of investigation.

Table of Contents

Introduction

CCDs suffer from a wide variety of effects which need to be calibrated and removed before scientific analysis can be completed, such as detector artifacts (hot pixels), deviations in light sensitivity (fringing), and detector noise sources (read-out noise, dark current) among others.

A recent paper by Boone et al., (2018; hereafter B18) outlines a potential new systematic error source for CCD imaging and spectroscopy, called the binary offset effect (BOE). The source of this uncertainty is related to the analog-to-digital conversion (ADC) process, which converts the analog electronic signal of the detector into a digital number as represented by binary bits. The BOE is due to cross-talk between the ADC for a source or driver pixel and pixels read out after the driver. B18 demonstrated that several CCD devices, including ACS and STIS, possess some evidence of a BOE in the serial readout direction.

Motivated by this result, the STIS team has conducted a preliminary investigation of the BOE for the STIS CCD. In the next Section, we reproduce the results of B18 using their publicly available scripts via GitHub and extend their analysis to datasets that span from 1997 to the present. Based on the results of our measurements, we also make some recommendations to users who might be concerned about BOE and when it may come into play above other noise sources on the CCD. For more details about the BOE, readers are referred to B18.

Binary Offset Measurements

B18 provided a Juptyer notebook for the scripts and commands that were used in their analysis of the BOE. Briefly, the script first takes an input 2-D image and determines a background that is either assumed constant (such as for a dark or bias image), or in principle could be fit with a 2-D function. The script then masks out any hot pixels or cosmic rays and determines all pixel values that fall within 2-σ of the mean of the image. The script then takes pixel values with at least 20 examples and calculates the sigma-clipped mean of the residuals from 1-3 pixels along the serial direction as well as the uncertainty of the residual. The BOE is present if the residuals exceed the estimated uncertainties and appear correlated to changes in the numbers of "1" bits in the binary representation of an analog-to-digital unit (ADU).

For the purposes of STIS, B18 analyzed a total of 72 bias files from the Cycle 24 calibration program 14820 (PI: A. Riley) which included a mixture of amplifiers. They highlighted three specific bias images in their paper (datasets odbn07030, odbn08030, and odbn1r030), all of which had an assumed constant background. These three highlighted biases used amplifier A and had GAIN=1. We note that Amplifier A is not currently used for scientific images and has a differing bias level at GAIN=1 compared to that of the amplifier D at GAIN=1. Amplifier D is used for all scientific exposures with STIS. B18 also noted that the BOE for the three biases appeared to change with time.

To further investigate this effect, we ran the scripts on 1328 other bias frames taken with amplifier D, GAIN=1. Roughly half of the biases are taken from the first year of STIS operations, 1997, with the other half obtained from 2018. Following the above time sampling, we also obtained 328 bias frames with GAIN=4, including 78 from a single anneal period between January and February 2018. Figure 1 shows our results for GAIN=1, while Figure 2 shows our results for GAIN=4. The typical uncertainty in our residuals is ~0.01 ADU for both GAIN=1 and GAIN=4.

Figure 1 shows that we recover a significant variation in signal residual with the number of "1" bits within a binary representation of a driver pixel's signal. At some signal levels there appears to be moderate to strong systematic differences, suggestive of a time variable component to this effect on STIS, as first reported by B18 for amplifier A. For GAIN=1, we find that the typical residual is peak-to-peak 0.5 ADU, with a maximum peak-to-peak residual of 2 ADU.

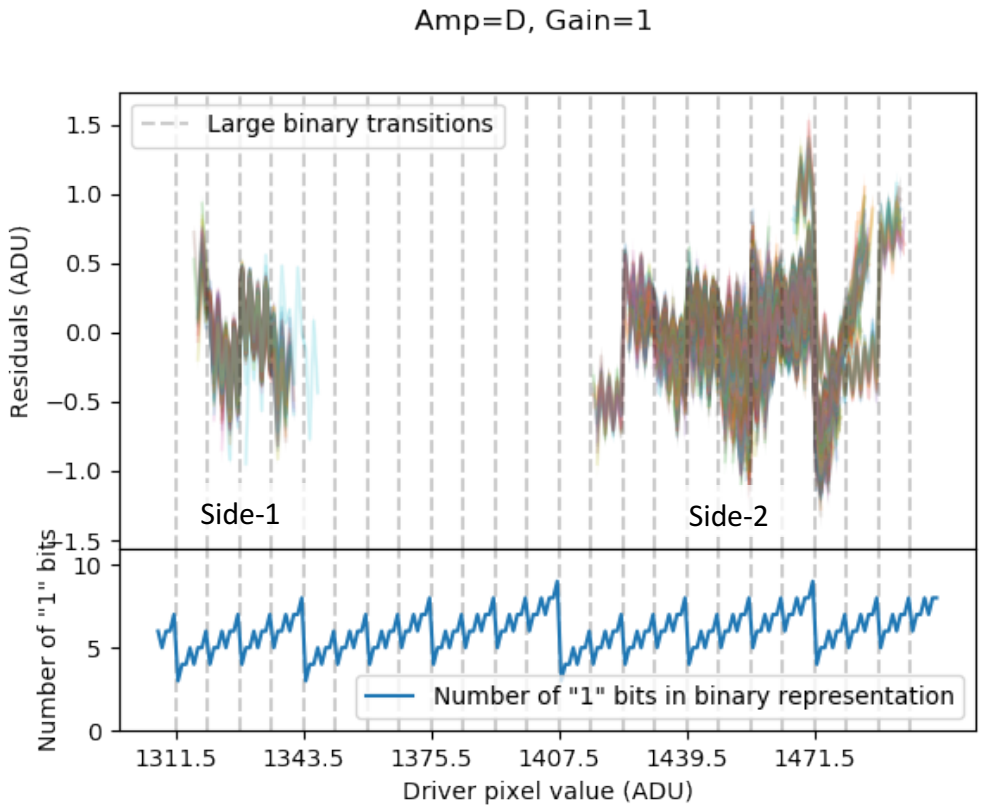

Figure 1: Amplifier D, GAIN=1 BOE measurements for a total of 1328 bias frames taken between 1997 and 2018. Different colors correspond to individual bias frames. There is a significant residual as a function of the number of binary "1" bits, similar to what B18 discovered in their analysis, albeit with a lower amplitude. Some evidence of time variability is present, especially near driver pixel values of 1471.5, which samples one of the larger jumps in the number of "1" bits.

In this figure, the lower signal levels correspond to detector operations during Side-1, while the higher signal levels correspond to detector operations with Side-2, when temperature control of the CCD was no longer possible. The relative stability of the lower signal pixels hints that variations due to temperature (or over differing anneal period) might explain the small systematic differences in the shape of the curves.

Figure 2 shows variation for the residuals with GAIN=4 but the relation is less clear due to significant variability over time, which is about as large as the amplitude of the BOE. The overall magnitude of the residuals is consistent with that of GAIN=1 when accounting for the factor of four difference between the two readout modes. To clarify the behavior at GAIN=4, we also measured the BOE over

a single anneal period. The results of that analysis are shown in Figure 3, showing that variability exists within a single anneal period as well.

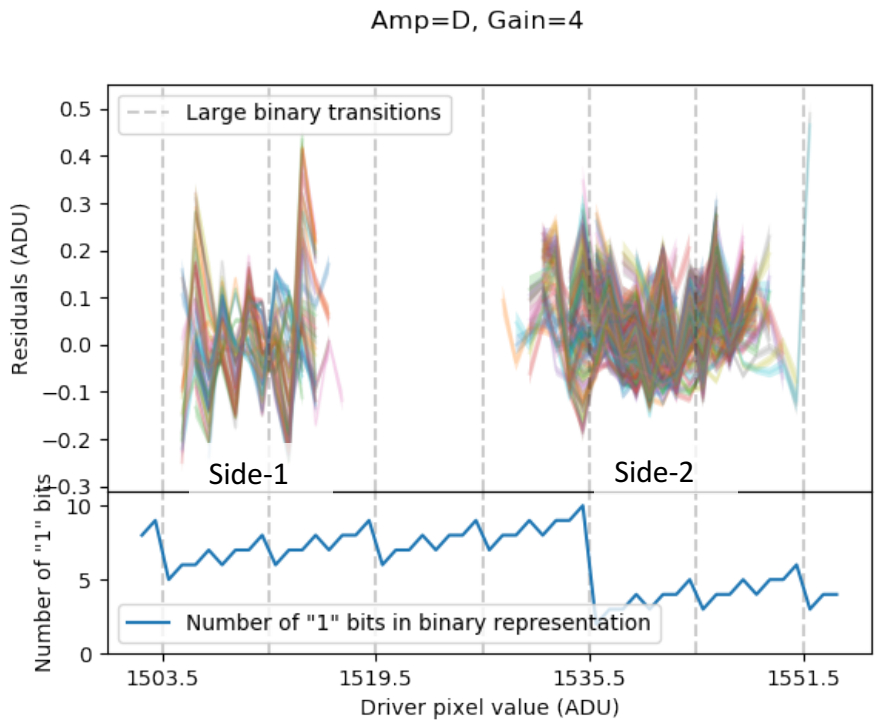

Figure 2: Same as for Figure 1, but for GAIN=4, from 328 bias frames. Here trends are less clear, potentially due to time variability of the BOE, or due to systematic effects that are not corrected for in the current analysis.

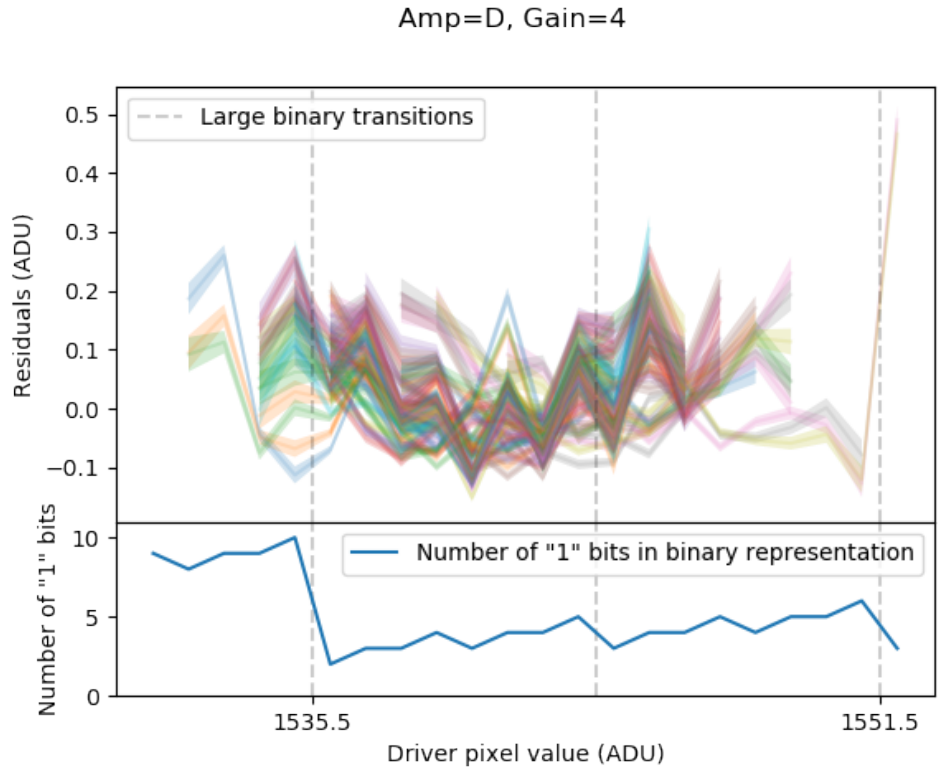

Figure 3: Same as for Figure 2 at GAIN=4, but for a smaller subset of 78 bias frames from a single annealing period in January-February, 2018. Even within a single anneal period there appears to be significant variability between bias frames.

Recommendations

We compare the magnitude of the systematic uncertainty with other noise sources present in the STIS CCD to gauge the severity of the BOE. Given the small level of the effect, we consider low SNR observations in particular, where the effect could do the most to degrade signal. In low SNR regimes and short exposure times, read noise dominates. Currently, the read noise of the STIS CCD at GAIN=1 is 6.24 ADU and in GAIN=4 it is 2.04 ADU. Both readout modes also suffer from the so-called herringbone effect (Brown, 2001; Goudfrooij et al., 2009), which adds ~1 ADU of noise to the total RMS read noise, and localized noise of several ADU peak-to-peak. Compared to these noise sources, the BOE is a minor perturbation. The BOE therefore appears to be an effect that can be ignored for the majority of science cases.

However, the BOE may be important for limited scientific cases currently used with STIS. For example, this effect may be important for the detection of narrow and weak absorption features next to bright emission lines, or for imaging programs that require accurate centroiding. In this case it might be desired to try to model the BOE following the procedure of B18. The STIS team encourages the astronomical community to further investigate whether BOE might be relevant to their individual science goals and to contact the STScI help desk[1] if any impacts are discovered.

Future work, if deemed of high enough priority, would be to better characterize why the STIS CCD BOE may vary systematically with time.

Conclusions

The STIS team has conducted a preliminary investigation of BOE for the STIS CCD. A significant effect is detected for typical scientific readout modes, in agreement with B18. Possible time variability of the effect is seen in STIS CCD biases using amplifier D with GAIN=1 and GAIN=4, which may complicate efforts to deterministically model and remove this effect. The magnitude of this systematic effect is likely small for most STIS science cases.

Acknowledgements

We thank K. Boone for helpful details regarding the STIS files used for their recent manuscript on the binary offset effect.

Change History for STIS ISR 2018-03

Version 1: 23 May 2018– Original Document

[1] https://stsci.service-now.com/hst

References

Boone, K. et al. 2018, PASP, 130, 988
Brown, T. 2001, STIS ISR 2001-05
Goudfrooij, P. et al. 2009, STIS ISR 2009-02

Appendix

For completeness, we include below a text rendering of the Binary Offset Effect measurement code used to generate Figures 1, 2, and 3 originally written by K. Boone, and with minor modifications by the STIS team to filter observation modes and to plot multiple imsets per file.

```
import sep
import glob
from astropy.io import fits
import numpy as np
from scipy.stats import sigmaclip
from scipy.signal import convolve2d
from matplotlib import pyplot as plt


def subtract_background(data):
    """Subtract the background from data.

    We assume that we are working with a bias or dark frame, so we don't
    have to worry about detecting and masking out objects.
    """
    background = sep.Background(data)

    sub_data = data - background.back()

    return sub_data


def build_cosmic_mask(data):
    """
    Build a cosmic ray mask for a residual image.

    This is a very naive algorithm. We are running on residuals, so we cut
    anything that is 5 sigma high or low. We then mask by a couple pixels
    around the cut too to capture the tail.
    """
    target_nmad = nmad(data)
    start_mask = np.abs(data - np.median(data)) > 5. * target_nmad

    num_edge_pixels = 2
    mask = ~(convolve2d(
        start_mask,
        np.ones((2 * num_edge_pixels + 1, 2 * num_edge_pixels + 1)),
        mode='same'
    ).astype(bool))

    return mask
```

```
def clipped_mean(data):
    clip_data, min_clip, max_clip = sigmaclip(data)

    mean = np.mean(clip_data)
    mean_err = np.std(clip_data) / np.sqrt(len(clip_data) - 1)

    return mean, mean_err


def nmad(data, *args, **kwargs):
    return 1.4826 * np.median(
        np.abs(np.asarray(data) - np.median(data, *args, **kwargs)),
        *args, **kwargs
    )


_cache_num_ones = {}


def cached_num_binary_ones(x):
    try:
        return _cache_num_ones[x]
    except KeyError:
        num_ones = _num_binary_ones(x)
        _cache_num_ones[x] = num_ones
        return num_ones


def _num_binary_ones(x):
    return bin(x).count('1')


def calc_num_binary_ones(x):
    x = np.atleast_1d(x)
    out = np.zeros(x.shape)
    for i in range(len(x.flat)):
        out.flat[i] = cached_num_binary_ones(x.flat[i])

    return out


def calculate_mean_residual(data, offset, do_diff=True, residuals=None):
    """Calculate the mean residuals as a function of the value of a
    reference pixel.

    This will return a list of raw pixel values and the average residuals
    for pixels separated by "offset" from pixels with those values.

    If do_diff is True, we look at a difference between the offset in the left
    direction and the right direction to cancel out effects like pickup.
    Otherwise, we only look in the direction specified by offset.

    The sign of offset doesn't really matter if used in diff mode since the
    positive and negative offsets will be subtracted from each other. However,
    if do_diff is False then they do matter. I have not determined the
    readout direction on each instrument tested here, so a positive offset may
    or may not be in the readout direction.
    """
    if residuals is None:
```

```
        # If we weren't given residuals, just assume that the image is relatively
        # flat and do a local background subtraction.
        residuals = subtract_background(data.astype(float).copy())

    # We look at the difference in residuals from one side to the next.
    ref_1 = np.roll(data, offset)
    ref_2 = np.roll(data, -offset)

    # Generate a mask to filter out cosmic rays/bad pixels.
    mask = build_cosmic_mask(data)
    use_residuals = residuals[mask]
    use_ref_1 = ref_1[mask]
    use_ref_2 = ref_2[mask]

    # Figure out which range of pixel values to probe.
    min_values = np.percentile(data, 2)
    max_values = np.percentile(data, 98)
    pixel_values = np.arange(min_values, max_values + 1)

    residual_means_1 = []
    residual_means_2 = []
    residual_errs_1 = []
    residual_errs_2 = []

    for pixel_value in pixel_values:
        cut_1 = use_ref_1 == pixel_value
        cut_2 = use_ref_2 == pixel_value

        # For each value that is found in at least 20 pixels we calculate a
        # clipped mean of the residuals and estimate an error on it.
        if np.sum(cut_1) > 20:
            mean_1, mean_err_1 = clipped_mean(use_residuals[cut_1])
        else:
            mean_1 = np.nan
            mean_err_1 = np.nan

        if np.sum(cut_2) > 20:
            mean_2, mean_err_2 = clipped_mean(use_residuals[cut_2])
        else:
            mean_2 = np.nan
            mean_err_2 = np.nan

        residual_means_1.append(mean_1)
        residual_means_2.append(mean_2)
        residual_errs_1.append(mean_err_1)
        residual_errs_2.append(mean_err_2)

    residual_means_1 = np.array(residual_means_1)
    residual_means_2 = np.array(residual_means_2)
    residual_errs_1 = np.array(residual_errs_1)
    residual_errs_2 = np.array(residual_errs_2)

    # Calculate the difference in residuals in each direction. This is what
    # we will look at to see if the binary offset effect is present since it
    # is relatively clean.
    if do_diff:
        residual_means = residual_means_1 - residual_means_2
        residual_errs = np.sqrt(residual_errs_1 ** 2 + residual_errs_2 ** 2)
    else:
        residual_means = residual_means_1
```

```
        residual_errs = residual_errs_1

    return pixel_values, residual_means, residual_errs


def setup_figure():
    fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True, gridspec_kw={'height_ratios':
        [2.5, 1], 'hspace': 0.})

    fig_parts = (fig, ax1, ax2)

    return fig_parts


def plot_mean_residual(data, offset, label=None, do_diff=True,
                       residuals=None, fig_parts=None):
    """Plot the mean residuals calculated using calculate_mean_residual."""
    pixel_values, residual_means, residual_errs = \
        calculate_mean_residual(data, offset, do_diff=do_diff, residuals=residuals)

    if fig_parts is not None:
        fig = fig_parts[0]
        ax = fig_parts[1]
    else:
        fig = plt.gcf()
        ax = fig.add_subplot(111)
        xlabel = 'Value of driver pixel (ADU)'
        ax.set_xlabel(xlabel)

        if offset == 1:
            offset_label = '1 pixel'
        else:
            offset_label = '%d pixels' % offset
        plt.ylabel('Mean residuals for an offset of %s (ADU)' % offset_label)

    ax.plot(pixel_values, residual_means, label=label)
    ax.fill_between(
        pixel_values,
        residual_means - residual_errs,
        residual_means + residual_errs,
        alpha=0.2
    )

    print('Median residual errors: %.4f' % np.median(residual_errs))

    return pixel_values


def finalize_figure(fig_parts, show_bits=4, min_bits=0, max_bits=11, legend_loc=0):
    fig, ax1, ax2 = fig_parts
    label_bit_changes(show_bits, ax1)
    labelled_locations = label_bit_changes(show_bits, ax2, legend=False)

    xmin, xmax = ax1.get_xlim()
    x = np.arange(int(np.floor(xmin)), int(np.ceil(xmax)))

    x_binary_count = calc_num_binary_ones(x)
    ax2.plot(x, x_binary_count, c='C0', label=\
        'Number of "1" bits in binary representation')
    ax2.set_ylabel('Number of "1" bits')
```

```
    ax2.set_xlabel('Driver pixel value (ADU)')
    ax2.set_ylim(min_bits, max_bits)
    ax2.legend(loc=legend_loc)

    ax2.set_xticks(labelled_locations)

    ax1.legend()


def calc_num_bits(x):
    return bin(int(x)).count('1')


def label_bit_changes(min_bits=4, axis=None, legend=True):
    if axis is None:
        axis = plt.gca()

    x_lim = plt.gca().get_xlim()
    x_start = int(np.floor(x_lim[0]))
    x_end = int(np.ceil(x_lim[-1]))

    x_range = np.arange(x_start, x_end)

    first = True
    labelled_locations = []
    for x in x_range:
        num_diff_bits = calc_num_bits(x ^ (x + 1))
        if num_diff_bits >= min_bits:
            if first and legend:
                label = 'Large binary transitions'
                first = False
            else:
                label = None
            axis.axvline(x + 0.5, c='black', ls='--', alpha=0.2, label=label)

            labelled_locations.append(x + 0.5)

    return labelled_locations


# STIS annealing period 2018-01-18 06:47:47 - 2018-02-13 16:14:23:
filenames = glob.glob('./data/single_anneal/*_raw.fits')

fig_parts = setup_figure()
means = []
temps = []
for filename in filenames:
    with fits.open(filename) as fits_file:
        if (fits_file[0].header['CCDAMP'] != 'D') | \
                (fits_file[0].header['CCDGAIN'] != 1) | \
                (fits_file[0].header['CCDOFFST'] != 3) | \
                (fits_file[0].header['SUBARRAY'] is True):
            continue
        for index in range(1, fits_file[0].header['NEXTEND'], 3):
            data = fits_file[index].data
            plot_mean_residual(data, 1, fig_parts=fig_parts)
            means.append(fits_file[index].header['GOODMEAN'])
            temps.append(fits_file[index].header['OCCDHTAV'])

finalize_figure(fig_parts)
```

```
means = np.array(means)
temps = np.array(temps)

fig_parts[0].suptitle('Amp=D, Gain=1')
fig_parts[0].savefig('stis_binaryoffset_D1.png')
```