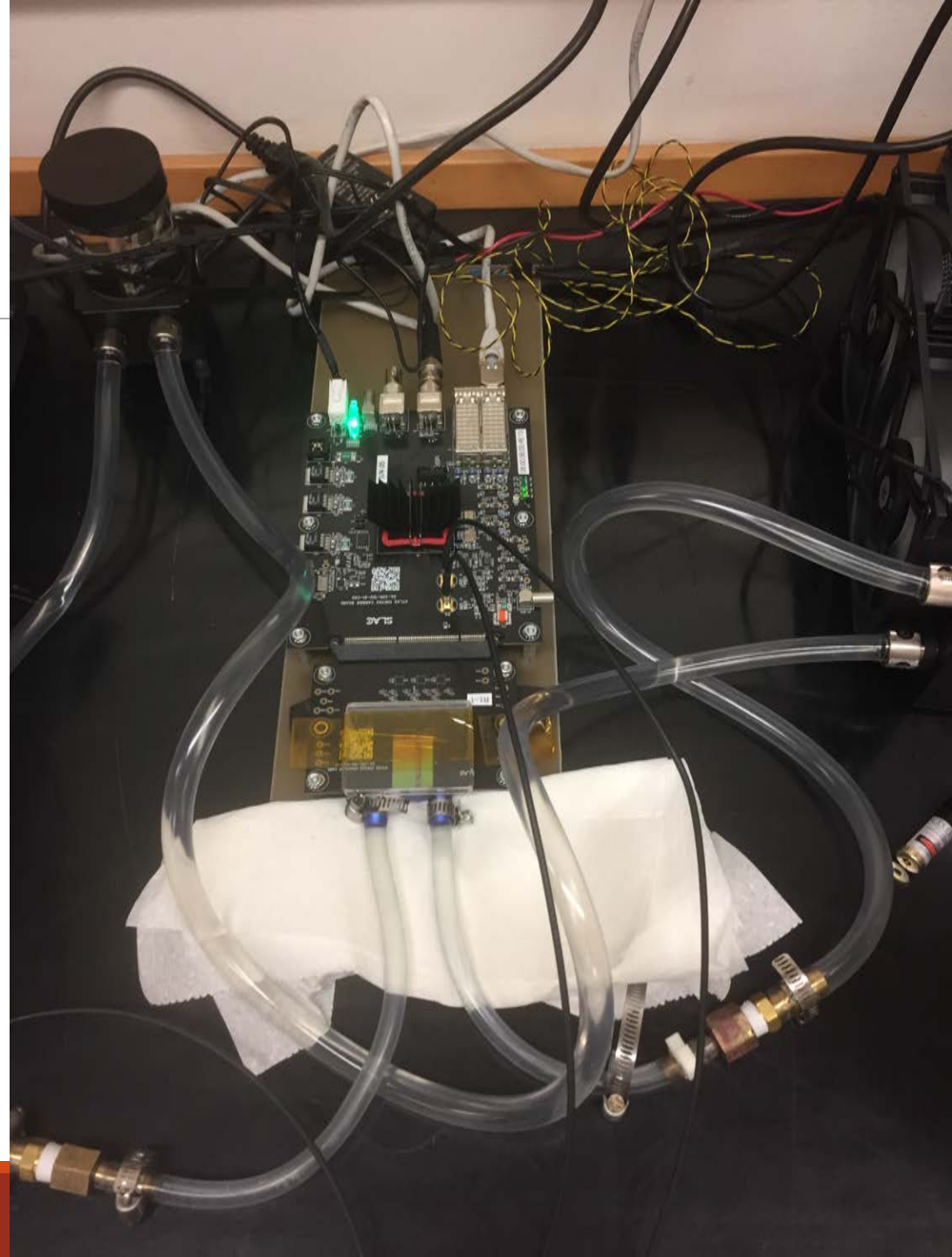
CHESS2 data acquisition

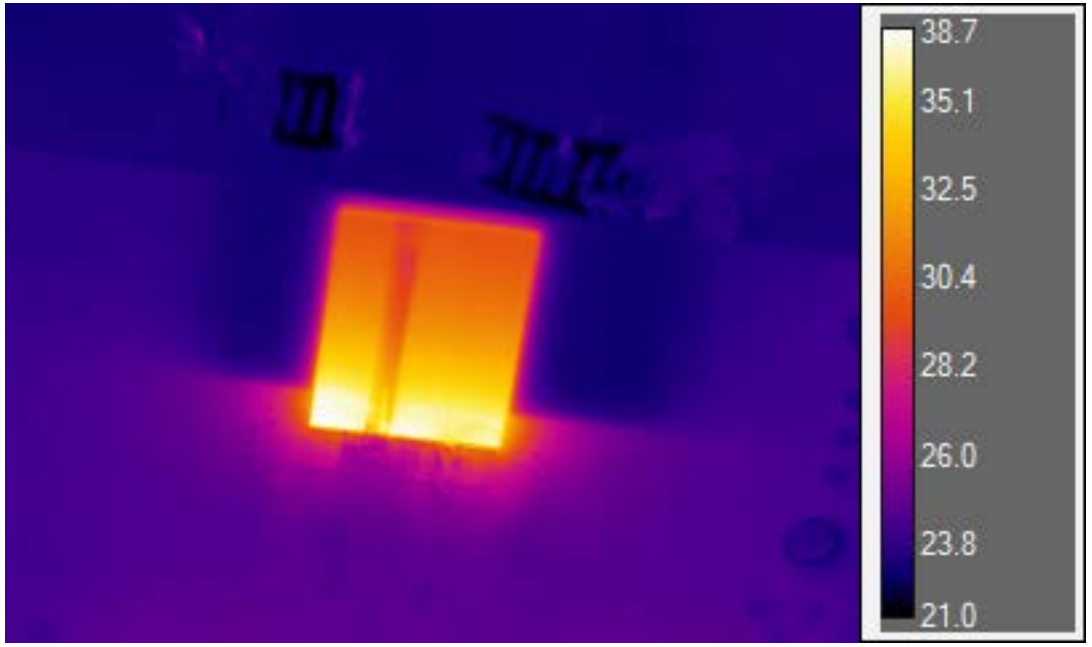
Setup

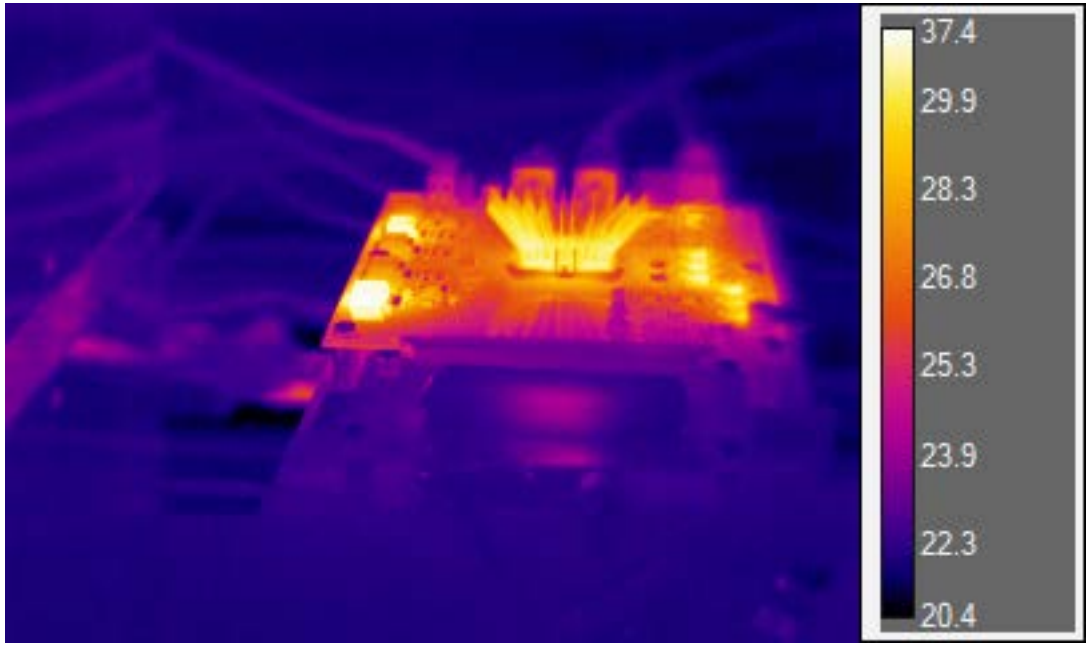
CHESS2 is water cooled using CPU cooling jig

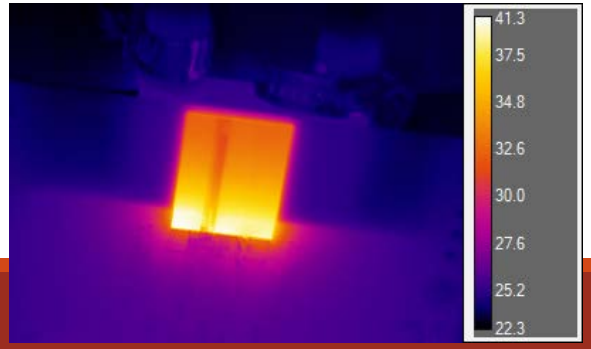
A cooling



Thermal picture







Data acquisition software

We implemented an OO framework

- Class HitMap_Plotter (plotting hitmap)
- Class Chess_control (configuring Chess2)
- Class Frame_data (to decode frame and process hitmaps)

Data acquisition software

Extended the _acceptFrame function

- Callback function executed each time a frame arrives
- Process frames on the fly
- Accumulates hitmap
- Counts number of frame
- Very fast

```
class EventReader(rogue.interfaces.stream.Slave):

    def __init__(self):
        rogue.interfaces.stream.Slave.__init__(self)
        self.plotter = Hitmap_Plotter()
        self.counter = 0
        self.ev_hitmap_t0 = np.zeros((128,32))
        self.ev_hitmap_t1 = np.zeros((128,32))
        self.ev_hitmap_t2 = np.zeros((128,32))

    def _acceptFrame(self,frame):
        p = bytearray(frame.getPayload())
        frame.read(p,0)
        f = Frame_data(p)
        f.decode_frame()
        self.hitmap_update(f)
        self.counter += 1
        print(self.counter)

    def hitmap_update(self,frame_data):
        self.ev_hitmap_t0 += frame_data.hitmap_t0
        self.ev_hitmap_t1 += frame_data.hitmap_t1
        self.ev_hitmap_t2 += frame_data.hitmap_t2

    def hitmap_show(self):
        self.plotter.show()

    def hitmap_plot(self):
        self.plotter.add_data(self.ev_hitmap_t0,
                              self.ev_hitmap_t1,
                              self.ev_hitmap_t2)
        self.plotter.plot()

    def hitmap_reset(self):
        self.counter = 0
        self.ev_hitmap_t0 = np.zeros((128,32))
        self.ev_hitmap_t1 = np.zeros((128,32))
        self.ev_hitmap_t2 = np.zeros((128,32))
```

Data acquisition software

New class Hitmap_plotter

- Realtime plotting of hitmaps
- Very fast ~100f/s possible
- Buffer the figure canvas
- Uses matplotlib.blit to render data

```
import matplotlib.pyplot as plt
import matplotlib
import numpy as np

class Hitmap_Plotter:
  def __init__(self):
    self.fig, (self.ax0,self.ax1,self.ax2) = plt.subplots(3,1)
    self.im0 = self.ax0.imshow(np.zeros((128,32)),cmap='jet', aspect='auto',vmin=0,vmax=1)
    self.im1 = self.ax1.imshow(np.zeros((128,32)),cmap='jet', aspect='auto',vmin=0,vmax=1)
    self.im2 = self.ax2.imshow(np.zeros((128,32)),cmap='jet', aspect='auto',vmin=0,vmax=1)
    self.background0 = self.fig.canvas.copy_from_bbox(self.ax0.bbox)
    self.background1 = self.fig.canvas.copy_from_bbox(self.ax1.bbox)
    self.background2 = self.fig.canvas.copy_from_bbox(self.ax2.bbox)

  def add_data(self,data0,data1,data2):
    self.data0 = data0
    self.data1 = data1
    self.data2 = data2

  def show(self):
    plt.pause(0.2)

  def plot(self):
    self.fig.canvas.restore_region(self.background0)
    self.fig.canvas.restore_region(self.background1)
    self.fig.canvas.restore_region(self.background2)
    self.im0.set_data(self.data0/max(1,np.max(self.data0)))
    self.im1.set_data(self.data1/max(1,np.max(self.data1)))
    self.im2.set_data(self.data2/max(1,np.max(self.data2)))
    self.ax0.draw_artist(self.im0)
    self.ax1.draw_artist(self.im1)
    self.ax2.draw_artist(self.im2)
    self.fig.canvas.blit(self.ax0.bbox)
    self.fig.canvas.blit(self.ax1.bbox)
    self.fig.canvas.blit(self.ax2.bbox)
```

Data acquisition software

New class Frame_data

- Defines frames objects
- Decoding methods implemented within

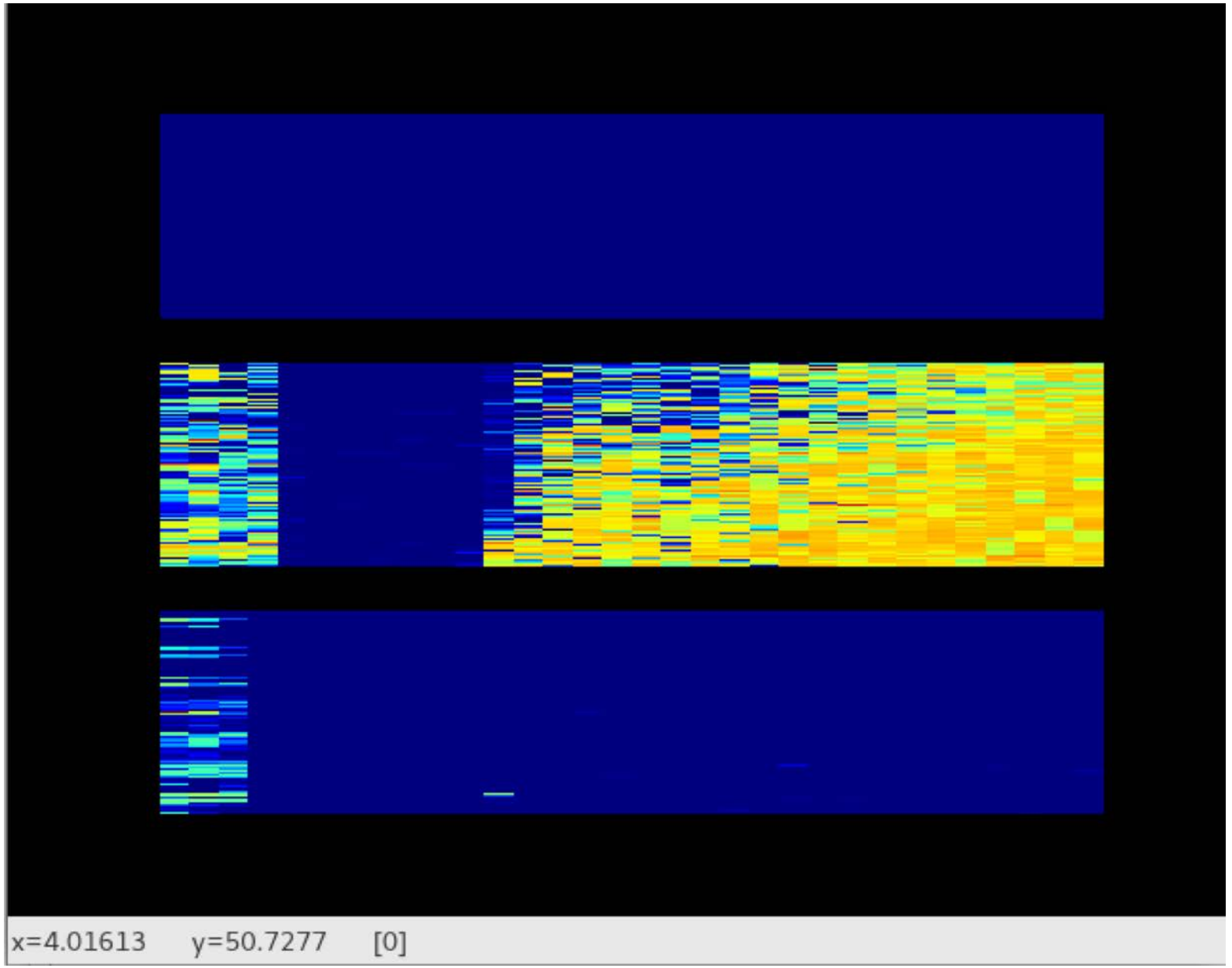
```
class Frame_data():
  def __init__(self,frame):
    self.frame = frame

    self.dvflag_M0=[]
    self.mhflag_M0=[]
    self.col_M0=[]
    self.row_M0=[]

    self.dvflag_M1=[]
    self.mhflag_M1=[]
    self.col_M1=[]
    self.row_M1=[]

    self.dvflag_M2=[]
    self.mhflag_M2=[]
    self.col_M2=[]
    self.row_M2=[]

    self.hitmap_t0=np.zeros((128,32))
    self.hitmap_t1=np.zeros((128,32))
    self.hitmap_t2=np.zeros((128,32))

  def __str__(self):
    msg = str("Hitmap 0 (self.hitmap_t0):"+str(self.hitmap_t0)+"\n")
    msg += str("Hitmap 1 (self.hitmap_t1):"+str(self.hitmap_t1)+"\n")
    msg += str("Hitmap 2 (self.hitmap_t2):"+str(self.hitmap_t2)+"\n")
    return msg

  @classmethod
  def decode_data_16b(cls,dat):
    row = dat[0] & 0x7F
    col = ((dat[1] & 0x0F) << 1) | (dat[0] >> 7)
    multi_hit  = (dat[1] & 0x10) >> 4
    data_valid = (dat[1] & 0x20) >> 5
    #dec_ok     = (dat[1] & 0xC0) >> 6
    return data_valid, multi_hit, col, row

  def decode_data(self,dat):
    if not all(d == 0 for d in dat): #To speed things up - most frames have empty data
      for i in range(0,len(dat),8):
```

HitMap collection

CHESS2 can only send 8 hits.

To produce an unbias hitmap we need to define windows of 8x1 enabled pixels and move it around.

1. Enable pixel for window of 1x8
2. Collect frame and accumulate hit map
3. Move to new window
4. Plot the hit map


x=4.01613 y=50.7277 [0]

Conclusion

Water cooling ~200$ (based on CPU water cooling solution) keep CHESS2 at a max of 40C

Data acquisition modified to be Object Oriented and processing frames on the fly. Works great but for very heavy frame rates plotting tends to hang (thinking of moving it to an other thread)

Data acquisition environment is ready for more advanced tests now.

- Understand the parameter space of Pixel configuration
- We have access to an alpha source.

We confirm the presence of a dead region in array 1. To be investigated.