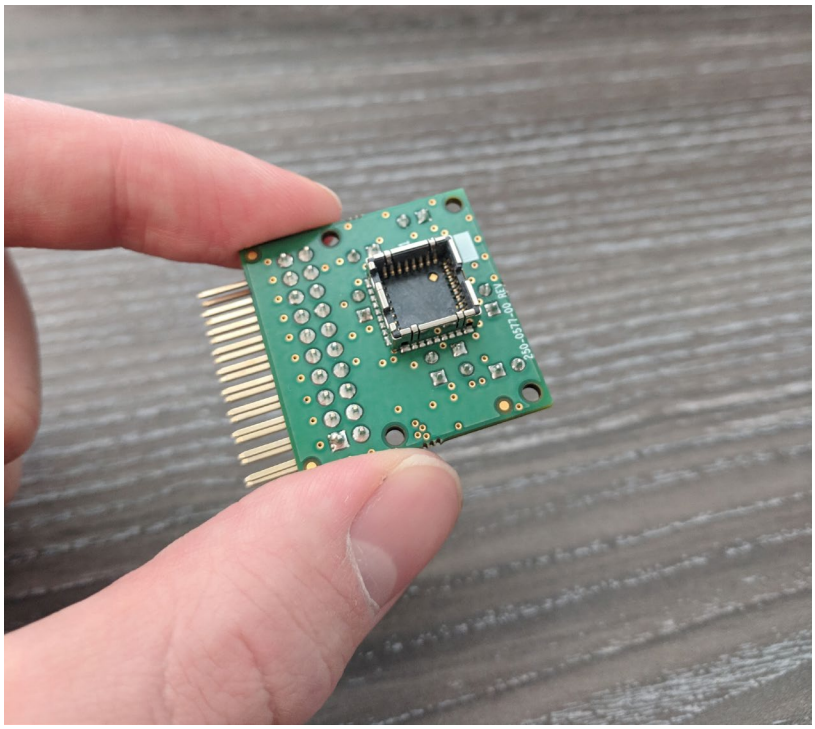
GETTING STARTED WITH THE BEAGLEBONE AND BREAKOUT BOARD V2.0

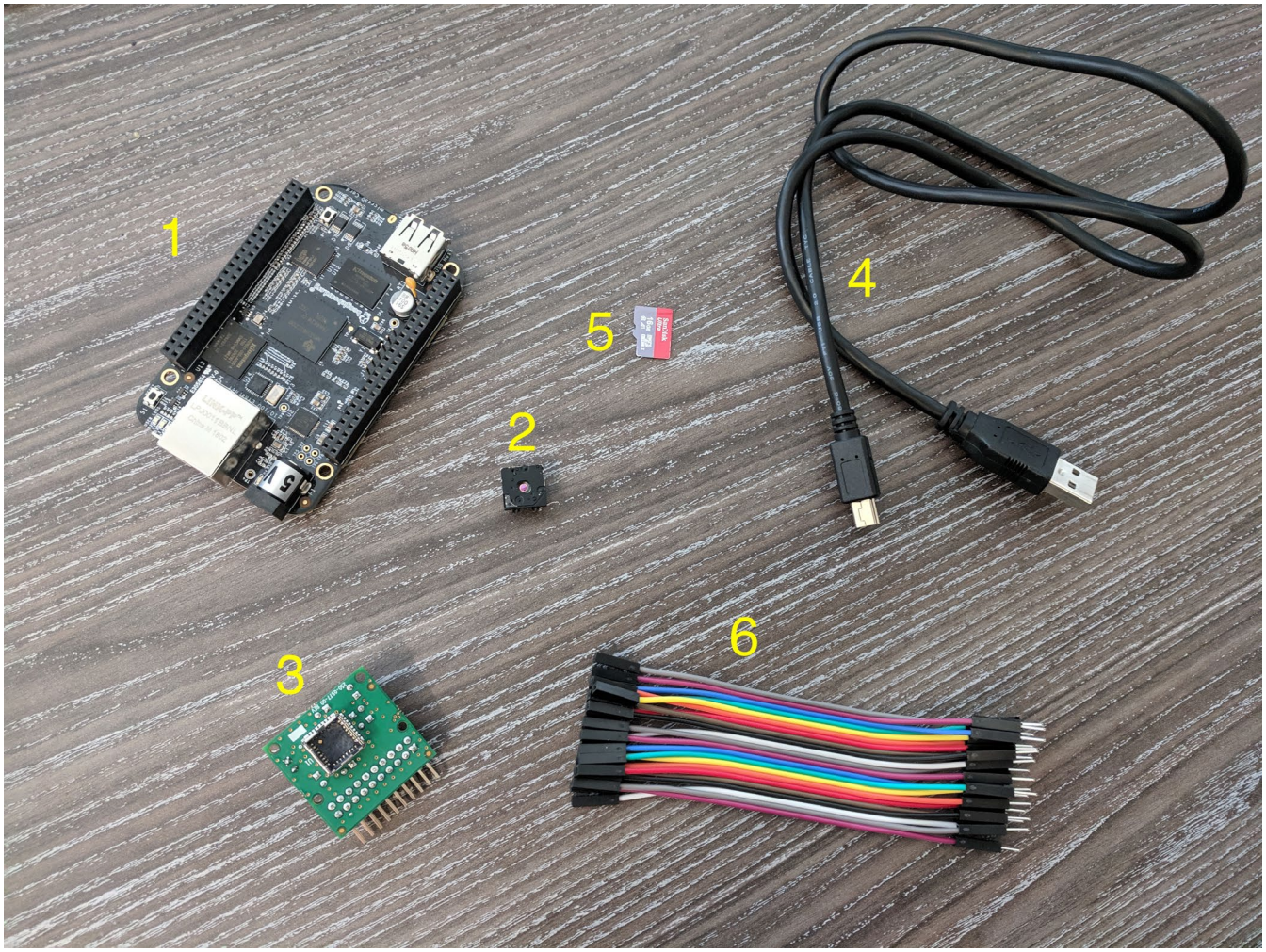
Disclaimer
This guide is for the newer breakout board (seen here). Earlier versions of the breakout board do not include VSYNC



Requirements

- A computer with the ability to read/write to MicroSD cards
    - A laptop using Linux was used during this guide.
- A BeagleBone
    - You can pickup a BeagleBone Black here.
    - Labelled as 1.
- A Lepton camera (version 2.X/3.X)
    - They can be found here. For a higher resolution and telemetry data, it is recommended that you use a Lepton 3.5.
    - Labelled as 2.

- A breakout board V2.0
    - Details on the board can be found here.
    - Labelled as 3.
- A MicroUSB to USB-A cable
    - Included with the BeagleBone from digikey.
    - Labelled as 4
- A MicroSD card
    - It should have a minimum capacity of 4GB
    - A 16GB one was used during this guide
    - Labelled as 5
- 8x Female-to-Male Jumper cables
    - Labelled as 6.
- Some additional software that can be found here.



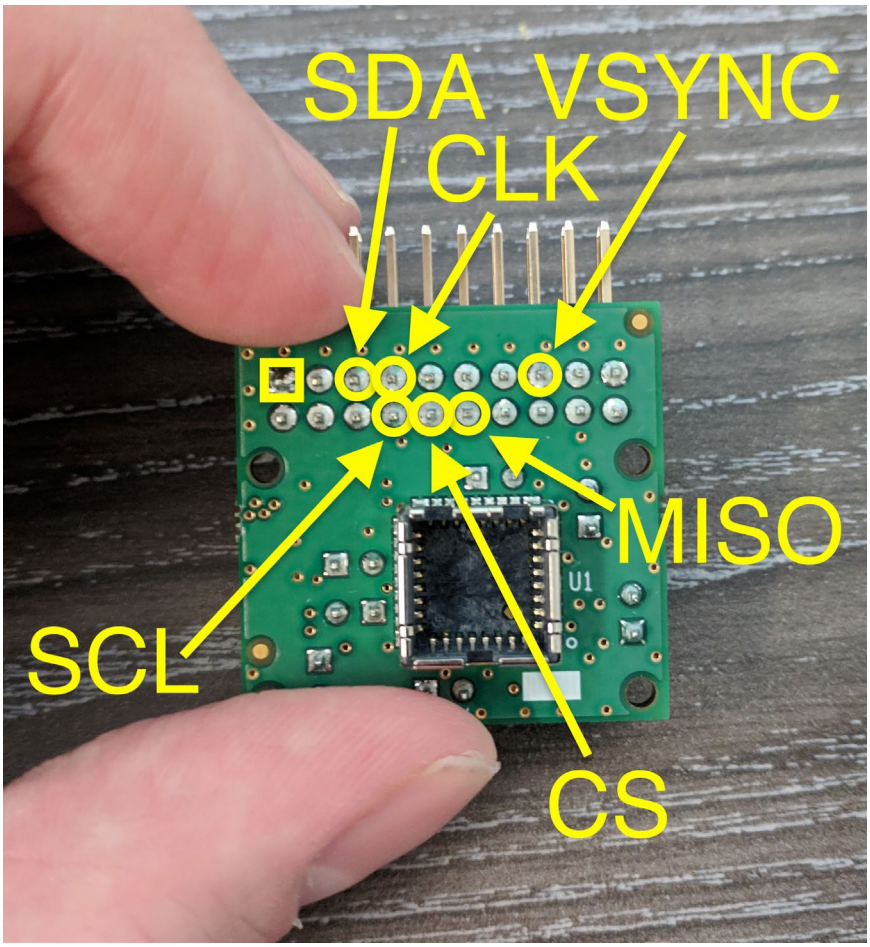
Hardware

Brand new BeagleBone

The BeagleBone will already have a function debian image installed on its eMMC, however for the purpose of this tutorial we'll be installing an additional debian image on a MicroSD. On boot, if it detects that there exists a MicroSD card with a debian image then it will boot from the card instead.

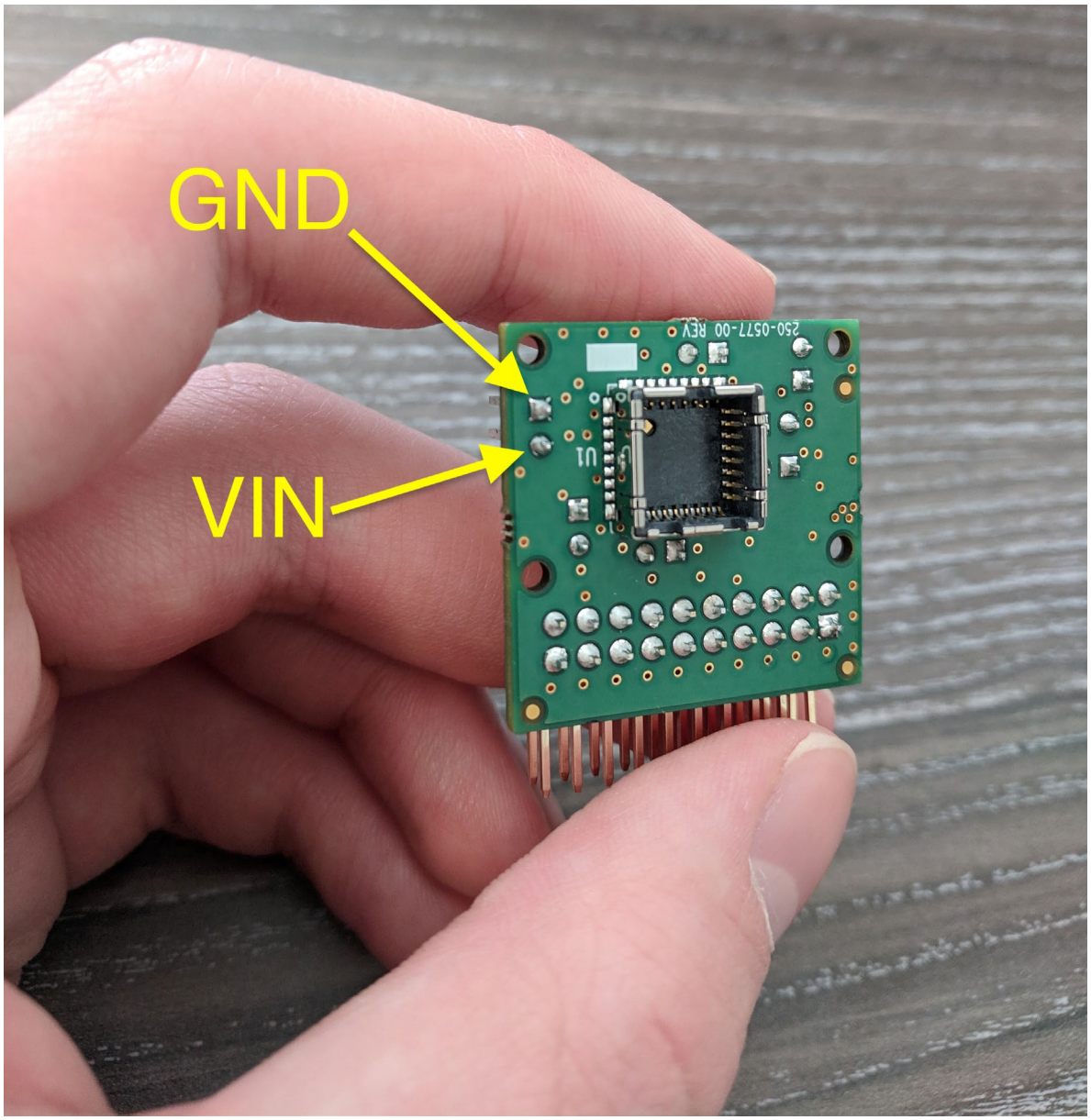
Attaching the breakout board to the BeagleBone

Attach (female-to-male) jumper wires between the following breakout board pins and the BeagleBone P9/P8 connector. You can find a link to the numbering of the BeagleBone's connectors here.



- (J2 Pin) -> (Proper name) -> (BeagleBone connector pin)
- P8 -> SCL -> P9 pin 19
- P5 -> SDA -> P9 pin 20
- P12 -> MISO -> P9 pin 21
- P7 -> CLK -> P9 pin 22
- P10 -> CS -> P9 pin 17
- P15 -> VSYNC -> P8 pin 17

We will also need to connect power and ground to the board:



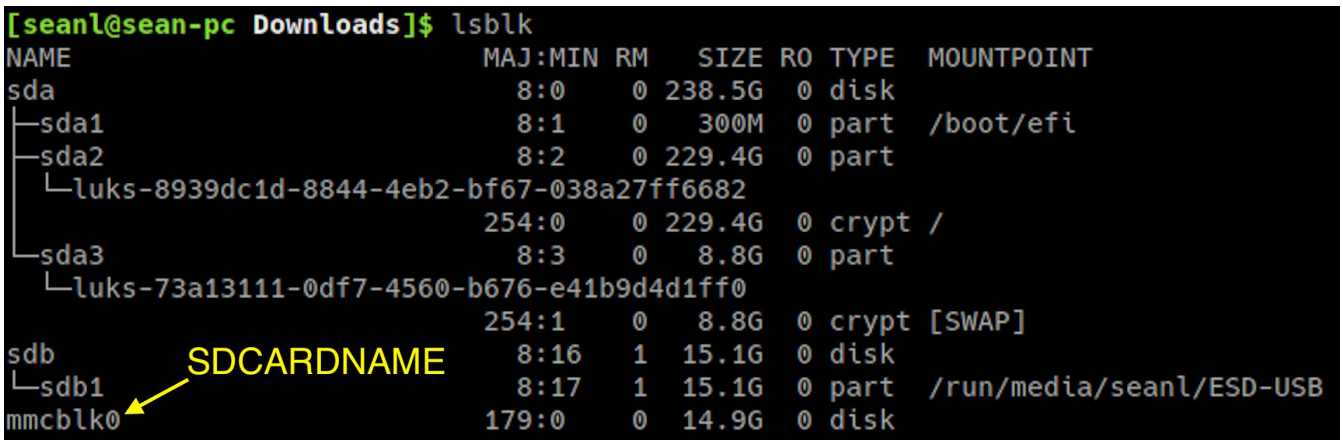
- (J3 Pin) -> (Proper name) -> (BeagleBone connector pin)
- P1 (Square pin) -> GROUND -> P9 pin 1
- P2 -> VIN -> P9 pin 3

Create the microSD card image
Load the desired image onto a microSD card.

- A 4GB IoT Debian Stretch image downloaded from https://beagleboard.org/latest-images has been tested and works.
- A permalink to the download that was used can be found here.
- If you wish to skip most of the installation steps and just be left with a BeagleBone Black with everything but the applications installed, then you can use one of the provided .images. They contain the correct Linux kernel, the device tree and the kernel module already setup.

Insert the MicroSD into your computer. You will need to find out the location of the SD card. The following command will give you that information:

```
lsblk
```



USING FRESH IOT DEBIAN STRETCH IMAGE

You will then need to xzcat the file you downloaded and then write it to the directory you just found. Assuming you downloaded the **permalinked image**:

```
xzcat bone-debian-9.5-iot-armhf-2018-10-07-4gb.img.xz | \
sudo dd bs=4M of=/dev/<SDCARDNAME> status=progress
```

There will be a few minutes of waiting while the MicroSD card is written to.

After the MicroSD has been written to, you will need to expand the partition size of the debian image to have enough room to build the kernel. Insert the MicroSD card into the BeagleBone and turn on the BeagleBone. The default user credentials for the Debian image are username - debian and password - temppwd. Once you have access to the terminal, run the following command:

```
sudo /opt/scripts/tools/grow_partition.sh
```

More information on expanding the partition can be found here.

Software

Installing the Toolchain and building the Linux kernel from source

You will need to build the Linux kernel with a specific configuration for the breakout board to properly work. This will require cloning a repository from github. We'll be building a specifc version of the kernel, 4.9.y:

```
git clone https://github.com/RobertCNelson/ti-linux-kernel-dev.git
cd ti-linux-kernel-dev/
git checkout ti-linux-4.9.y
```

Running build_kernel.sh will acquire the dependencies and eventually open a menu where you can adjust the config options.

```
./build_kernel.sh
```

You will need to ensure the following options are set:

- Due to low-latency requirements for collecting VOSPI data from the Lepton (missing subframe deadlines results in frame loss), disable the CPU idle functionality (unset CONFIG_CPU_IDLE in the kernel .config).
  - This change results in much lower frame loss, though it will result in higher overall power usage, if this is a concern.
- Verify that the DMA engine is enabled (CONFIG_DMA_ENGINE=y)
- Ensure that the VIDEBUF2 code is set:
  - CONFIG_VIDEOBUF2_CORE=m,
  - CONFIG_VIDEOBUF2_MEMOPS=m,
  - CONFIG_VIDEOBUF2_DMA_CONTIG=m,
  - CONFIG_VIDEOBUF2_VMALLOC=m
- (These can also be set to =y, to skip the requirement to load these modules before loading the Lepton module).

Additional instructions for building the kernel can found here.

Once we have We will now need to setup the new kernel. There is a script provided that will set everything up for you.

Make sure that the MicroSD is inserted into your machine. You will need to find out the location of the SD card. The following command will give you that information:

```
lsblk
```

Now that we have the location, you will need to setup the script. It can be found in ti-linux-kernel-dev/system.sh. If the file isn't there, then the build_kernel.sh script was not successful. You will need to edit the line:

```
#MMC=/dev/sde
```

to

```
MMC=/dev/SDCARDNAME
```

where SDCARDNAME is the location of the MicroSD card in /dev.

Then from the same folder the script is located, run the following command:

```
./tools/install_kernel.sh
```

Once it has been succesfully installed, insert the MicroSD into the BeagleBone. Once it is up and running, run the following command:

```
uname -r
```

Ensure that the output matches the <version string> for the kernel you compiled. As an example, the version I get is 4.9.126-ti-r115.

Building the driver and applications

Now, from your development machine lets build the driver and test applications. First, set up the environment so that the correct toolchain and kernel source directory can be found.

```
export PATH=$PATH:[path to ti-linux-kernel-dev]/dl/gcc-linaro.../bin
export CROSS_COMPILE=arm-linux-gnueabihf-
export KDIR=[path to ti-linux-kernel-dev]/KERNEL
```

(Replace gcc-linaro… with the directory name found under dl/).

Next, build the I2C application that commands the Lepton to begin transmitting VSYNC pulses from its GPIO3 output and the user-space data collector application by running make from the top-level directory.

```
cd BeagleBone/
make
```

Next, build the kernel module:

```
cd BeagleBone/lepton_module/
make
```

We will then need to move the following files over to the MicroSD card:

- BeagleBone/lepton_control/vsync_app
- BeagleBone/lepton_module/lepton.ko
- BeagleBone/lepton_data_collector/lepton_data_collector
- BeagleBone/lepton_module/flir-lepton-00A0.dts

For the purpose of the tutorial, I'll be putting them on the MicroSD card in a folder located at ~/lepton-files.

Setting up device tree and kernel module

We'll be operating from the BeagleBone for the next steps, so ensure that the MicroSD is inserted into the BeagleBone, it is running and that you can access those files.

Lets ensure that the device file tree is ready. Beagleboard provides a script that makes the process very simple when run on the board:

```
git clone https://github.com/beagleboard/bb.org-overlays
cd bb.org-overlays/src/arm

# Copy the flir-lepton-00A0.dts file into here so that it
# is compiled and moved to the appropriate folders.
cp ~/lepton-files/flir-lepton-00A0.dts .
cd ../..

# Builds and installs the device tree overlays
./install.sh
```

Next, we'll need to point to this file in /boot/uEnv.txt. The device tree bindings should also be added to this file. Edit a line in the Additional custom capes section, and change E.G. <file4> to flir-lepton-00A0.

For example, change:

```
#uboot_overlay_addr4=/lib/firmware/<file4>.dtbo
```

to:

```
uboot_overlay_addr4=/lib/firmware/flir-lepton-00A0.dtbo
```

We will also need to move the kernel object into the appropriate location for it to be eventually loaded via modprobe.

```
sudo su
mkdir /lib/modules/`uname -r`/extra
cp home/debian/lepton-files/lepton.ko /lib/modules/`uname -r`/extra
```

For these changes to take effect, restart your BeagleBone.

```
sudo reboot
```

Finally, we'll build the kernel objects dependencies and install the kernel module.

```
sudo su
depmod -a
modprobe lepton dyndbg=+p
```

We should be set! Lets test if this worked.

Testing the lepton camera

We will need to run the vsync_app, to ensure that the Lepton is sending VSYNC pulses.

```
~/lepton-files/vsync_app
```

You can use the lepton_data_collector application to capture grayscale images from the video device.

- It consumes raw subframes via the V4L2 /dev/videoN device file (/dev/video0 by default) and extracts the pixel data.
- For Lepton 3.X (with the command-line argument -3) it assembles four subframes into a single larger video frame.
- The image files are named after a prefix specified with the -o command-line argument, followed by a 6-digit (prefixed with 0's for smaller numbers) frame counter and a .gray extension.
- To reduce latency, it is a good idea to have it store frames into memory instead of to a flash device, so mount a tmpfs directory somewhere and prepend the path to the prefix.

Here is an example:

```
cd ~/lepton-files
sudo su
mkdir /tmp/capture
mount -t tmpfs tmpfs /tmp/capture
./lepton_data_collector -3 -c 50 -o /tmp/capture/frame_
```

When complete, there should be 50 images captured from Lepton 3.X (about 5 seconds worth at ~9 fps) named /tmp/capture/frame_000000.gray through /tmp/capture/frame_000049.gray. These can be viewed on a Linux system using the ImageJ Java application, available from here. From the File menu, select Import->Raw..., and set image type to 16-bit unsigned along with the width and height (80x60 for Lepton 2.X, 160x120 for Lepton 3.X) in the dialog box that appears after selecting the file name. The entire sequence can be placed into an image stack if the Open all files in folder checkbox is checked. Loading the files into an image stack makes it possible to produce a .avi movie from the frames using File->Save As->AVI... and setting the frame rate to 9 fps.

Troubleshooting

Before troubleshooting

Additional debug information can be retrieved by running:

```
dmesg | less
```

"I've completed all the steps but for some reason the device tree isn't being read!"

Your old bootloader in the eMMC is blocking u-boot overlays, you can fix it by running following command from the BeagleBone:

```
sudo dd if=/dev/zero of=/dev/mmcblk1 bs=1M count=10 status=progress
```