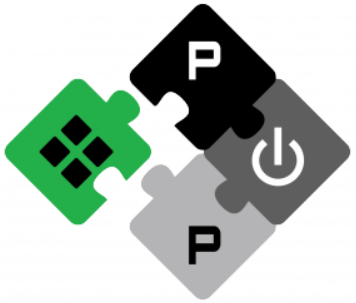


PULP PLATFORM
Open Source Hardware, the way it should be!

Bitcraze Workshop: Hands-on Session 2
Image acquisition and parallel image filter

Lorenzo Lamberti, *Hanna Müller*, Vlad Niculescu, Manuele Rusci, Daniele Palossi


http://pulp-platform.org


@pulp_platform


https://www.youtube.com/pulp_platform

The AI-Deck

Crazyflie + AI-Deck

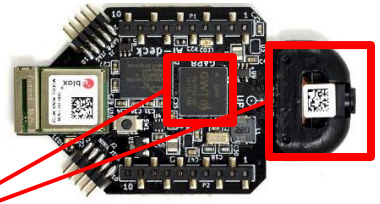
Crazyflie (STM32)

Radio: Nordic BTLE

nRF51 2.4GHz
Data rate: 0,25/1/2 Mbit/s

UART Link

Data rate: 1 Mbit/s



AI-Deck (GAP8)

Radio: NINA Wi-Fi

NINA-W102 2.4 GHz
Data rate: 6-54 Mbit/s

Radio dongle

Wi-Fi card

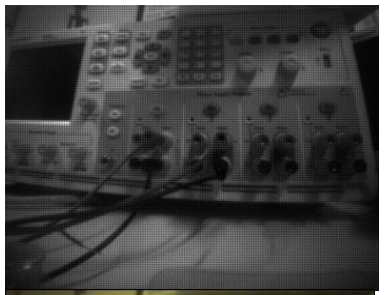
Hands-on 2: GAP8 programming & camera

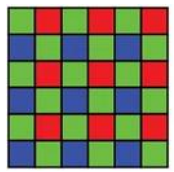
Hands-on: Image acquisition and filtering

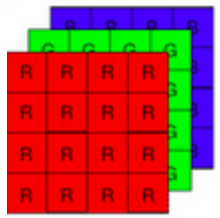
1. git clone https://github.com/bitcraze/AIdeck_examples
2. set up your gap-sdk (source configs/ai_deck.sh)
3. Go to GAP8/image_processing_examples/simple_kernel_example
4. Compile and run the code (make clean all run platform=board or gap_run in the VM)
5. You can configure some flags in the Makefile

First: execution flow using demosaicking on the fabric controller as example
Then: parallelization with inverting an image on the cluster.

The code is simplified on the slides (but functional)







Demosaicking Fabric controller

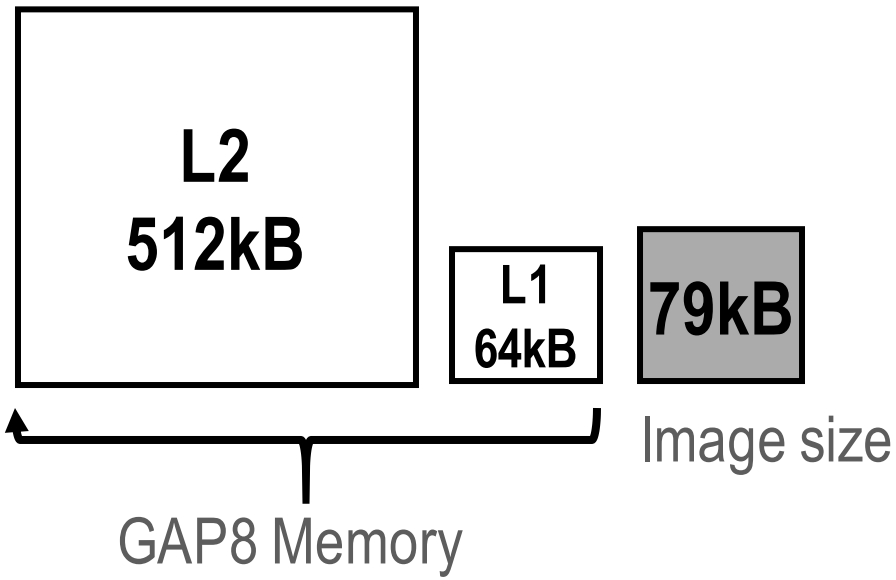
Inverting Fabric controller

Demosaicking Cluster

Inverting Cluster

Hands-on: Image acquisition and filtering

Before we start, let's think about memory:
How many QVGA images could you have on GAP8 at the same time?
Does it matter if they are colored or grey? Hint:
GAP8 L2 Memory:512kB



Not even a single grey scale one on L1.
6 grey scale or 2 RGB in L2 – BUT do not forget, you also need space for the code in L2!

Hands-on: Image acquisition and filtering

```
#include "pmsis.h"
#include "bsp/bsp.h"
#include "bsp/camera.h"
#include "bsp/camera/himax.h"

#include "gaplib/ImgIO.h"

#include "img_proc.h"

#define WIDTH    324
#ifdef QVGA_MODE
#define HEIGHT   244
#else
#define HEIGHT   324
#endif
#define BUFF_SIZE (WIDTH*HEIGHT)

PI_L2 unsigned char *buff;

PI_L2 unsigned char *buff_demosaick;

static struct pi_device camera;
static volatile int done;
```

Include drivers

Include image IO library

Include own demosaicking function

Define acquisition size

Define variables – place buffer in L2

```
static void handle_transfer_end(voic
{
    done = 1;
}

static int open_camera(struct pi_dev
{
    printf("Opening Himax camera\n")
    struct pi_himax_conf cam_conf;
    pi_himax_conf_init(&cam_conf);

#if defined(QVGA_MODE)
    cam_conf.format = PI_CAMERA_QVGA
#endif

    pi_open_from_conf(device, &cam_c
    if (pi_camera_open(device))
        return -1;
    pi_camera_control(device, PI_CAM

    return 0;
}
```

```
#ifdef ASYNC_CAPTURE
// Start up async capture task
done = 0;
pi_task_t task;
pi_camera_capture_async(&camera, buff, BUFF_SIZE, pi_task_callback(&task, handle_transfer_end, NULL));
#endif
```

```
APP = test
APP_SRCS += test.c $(GAP_LIB_PATH)/img_io/ImgIO.c img_proc.c
APP_INC  += . $(GAP_LIB_PATH)/include

APP_CFLAGS += -O3 -g


PMSIS_OS ?= pulp_os

APP_CFLAGS += -DASYNC_CAPTURE
APP_CFLAGS += -DQVGA_MODE
APP_CFLAGS += -DCOLOR_IMAGE


clean::
    rm -rf img_raw.ppm img_color.ppm img_gray.ppm

include $(RULES_DIR)/pmsis_rules.mk
```

```
{
    printf("\n\t*** PMSIS Camera Example ***\n\n");
    return pmsis_kickoff((void *) test_camera);
}
```


Hands-on: Image acquisition and filtering

Include drivers

Include image IO library

Include own demosaicking function

Define acquisition size

Define variables – place buffer in L2

```c
#include "pmsis.h"
#include "bsp/bsp.h"
#include "bsp/camera/himax.h"

#define WIDTH    324
#ifdef QVGA_MODE
#define HEIGHT   244
#else
#define HEIGHT   324
#endif

PI_L2 unsigned char *buff;

PI_L2 unsigned char *buff_demosaick;

static struct pi_device camera;
static volatile int done;

static void handle_transfer_end(void *arg)
{
    done = 1;
}

static int open_camera(struct pi_device *device)
{
    printf("Opening Himax camera\n");
    struct pi_himax_conf cam_conf;
    pi_himax_conf_init(&cam_conf);

#if defined(QVGA_MODE)
    cam_conf.format = PI_CAMERA_QVGA;
#endif

    pi_open_from_conf(device, &cam_conf);
    if (pi_camera_open(device))
        return -1;
    pi_camera_control(device, PI_CAMERA_CMD_AEG_INIT, 0);

    return 0;
}
```

```c
int test_camera()
{
    printf("Entering main controller\n");

    #ifdef ASYNC_CAPTURE
    printf("Testing async camera capture\n");

    #else
    printf("Testing normal camera capture\n");
    #endif

    // Open the Himax camera
    if (open_camera(&camera))
    {
        printf("Failed to open camera\n");
        pmsis_exit(-1);
    }


    // Rotate camera orientation
    uint8_t set_value=3;
    uint8_t reg_value;

    pi_camera_reg_set(&camera, IMG_ORIENTATION, &set_value);
    pi_camera_reg_get(&camera, IMG_ORIENT
    printf("img orientation %d\n",reg_val

    #ifdef QVGA_MODE
    set_value=1;
    pi_camera_reg_set(&camera, QVGA_WIN_E
    pi_camera_reg_get(&camera, QVGA_WIN_E
    printf("qvga window enabled %d\n",reg
    #endif

    #ifndef ASYNC_CAPTURE
    set_value=0;
    pi_camera_reg_set(&camera, VSYNC_HSYNC_PIXEL_SHIFT_EN, &set_value)
    pi_camera_reg_get(&camera, VSYNC_HSYNC_PIXEL_SHIFT_EN, &reg_value)
    printf("vsync hsync pixel shift enabled %d\n",reg_value);
    #endif

    // Reserve buffer space for image
    buff = pmsis_l2_malloc(BUFF_SIZE);
    if (buff == NULL){ return -1;}

    #ifdef COLOR_IMAGE
    buff_demosaick = pmsis_l2_malloc(BUFF_SIZE*3);
    #else
    buff_demosaick = pmsis_l2_malloc(BUFF_SIZE);
    #endif
    if (buff_demosaick == NULL){ return -1;}
    printf("Initialized buffers\n");
```

```c
    #ifdef ASYNC_CAPTURE
    // Start up async capture task
    done = 0;
    pi_task_t task;
    pi_camera_capture_async(&camera, buff, BUFF_SIZE, pi_task_callback(&task, handle_transfer_end, NULL));
    #endif

    // Start the camera
    pi_camera_control(&camera, PI_CAMERA_CMD_START, 0);
    #ifdef ASYNC_CAPTURE
    while(!done){pi_yield();}
    #else
    pi_camera_capture(&camera, buff, BUFF_SIZE);
    #endif

    // Stop the camera and immediately close it
    pi_camera_control(&camera, PI_CAMERA_CMD_STOP, 0);
    pi_camera_close(&camera);


    #ifdef COLOR_IMAGE
    demosaicking(buff, buff_demosaick, WIDTH, HEIGHT, 0);
    #else
    demosaicking(buff, buff_demosaick, WIDTH, HEIGHT, 1);
```

```c
int main(void)
{
    printf("\n\t*** PMSIS Camera Example ***\n\n");
    return pmsis_kickoff((void *) test_camera);
}
```

Set up OS, then jump to test_camera

Hands-on: Image acquisition and filtering

Include drivers

Include image IO library

Include own demosaicking function

Define acquisition size

Define variables

```c
#include "pmsis.h"
#include "bsp/bsp.h"
#include "bsp/camera/himax.h"

#define WIDTH    324
#ifdef QVGA_MODE
#define HEIGHT   244
#else
#define HEIGHT   324
#endif
#define BUFF_SIZE (WIDTH*HEIGHT)

PI_L2 unsigned char *buff;

PI_L2 unsigned char *buff_demosaick;

static struct pi_device camera;
static volatile int done;
```

```c
static void ha
{
    done = 1;
}
```

```c
static int ope
{
    printf("Op
    struct pi
    pi_himax_c

#if defined(QVG
    cam_conf.fo
#endif

    pi_open_fr
    if (pi_came
        return
    pi_camera_

    return 0;
}
```

Open and initialize camera

```c
static int open_camera(struct pi_device *device)
{
    printf("Opening Himax camera\n");
    struct pi_himax_conf cam_conf;
    pi_himax_conf_init(&cam_conf);

#if defined(QVGA_MODE)
    cam_conf.format = PI_CAMERA_QVGA;
#endif

    pi_open_from_conf(device, &cam_conf);
    if (pi_camera_open(device))
        return -1;
    pi_camera_control(device, PI_CAMERA_CMD_AEG_INIT, 0);

    return 0;
}
```

```c
int test_camera()
{
    printf("Entering main controller\n");

    #ifdef ASYNC_CAPTURE
    printf("Testing async camera capture\n");

    #else
    printf("Testing normal camera capture\n");
    #endif

    // Open the Himax camera
    if (open_camera(&camera))
    {
        printf("Failed to open camera\n");
        pmsis_exit(-1);
    }

    // Rotate camera orientation
    uint8_t set_value=3;
    uint8_t reg_value;

    pi_camera_reg_set(&camera, IMG_ORIENTATION, &se
    pi_camera_reg_get(&camera, IMG_ORIENTATION, &re
    printf("Camera orientation reg: %d\n", reg_value);
```

```c
    #ifdef ASYNC_CAPTURE
    // Start up async capture task
    done = 0;
    pi_task_t task;
```

Open camera

Hands-on: Image acquisition and filtering

Include drivers

Include image IO library

Include own demosaicking function

```
#include "pmsis.h"
#include "bsp/bsp.h"
#include "bsp/camera/himax.h"

#define WIDTH    324
#ifdef QVGA_MODE
#define HEIGHT   244
#else
#define HEIGHT   324
#endif
#define BUFF_SIZE (WIDTH*HEIGHT)

PI_L2 unsigned char *buff;

PI_L2 unsigned char *buff_demosaick;

static struct pi_device camera;
static volatile int done;

static void handle_transfer_end(void *arg)
{
    done = 1;
}

static int open_camera(struct pi_device *device)
{
    printf("Opening Himax camera\n");
    struct pi_himax_conf cam_conf;
    pi_himax_conf_init(&cam_conf);

#if defined(QVGA_MODE)
    cam_conf.format = PI_CAMERA_QVGA;
#endif

    pi_open_from_conf(device, &cam_conf);
    if (pi_camera_open(device))
        return -1;
    pi_camera_control(device, PI_CAMERA_CMD_AEG_INIT, 0);
```

Define acquisition size

Define variables – place buffer in L2

Open and initialize camera

```
int test_camera()
{
    printf("Entering main controller\n");

    #ifdef ASYNC_CAPTURE
    printf("Testing async camera capture\n");

    #else
    printf("Testing normal c
    #endif

    // Open the Himax camera
    if (open_camera(&camera)
    {
        printf("Failed to op
        pmsis_exit(-1);
    }
```

Open camera

```
    // Rotate camera orienta
    uint8_t set_value=3;
    uint8_t reg_value;

    pi_camera_reg_set(&camer
    pi_camera_reg_get(&camer
    printf("img orientation

    #ifdef QVGA_MODE
    set_value=1;
    pi_camera_reg_set(&camer
    pi_camera_reg_get(&camer
    printf("qvga window enab
    #endif

    #ifndef ASYNC_CAPTURE
    set_value=0;
    pi_camera_reg_set(&camer
    pi_camera_reg_get(&camer
    printf("vsync hsync pixe
    #endif
```

```
    // Reserve buffer space
    buff = pmsis_l2_malloc(B
    if (buff == NULL){ retur

    #ifdef COLOR_IMAGE
    buff_demosaick = pmsis_l
    #else
    buff_demosaick = pmsis_l
    #endif
    if (buff_demosaick == NU
    printf("Initialized buffers\n");
```

```
    #ifdef ASYNC_CAPTURE
    // Start up async capture task
    done = 0;
    pi_task_t task;
    pi_camera_capture_async(&camera, buff, BUFF_SIZE, pi_task_callback(&task, handle_transfer_end, NULL));
    #endif
```

```
    // Rotate camera orientation
    uint8_t set_value=3;
    uint8_t reg_value;

    pi_camera_reg_set(&camera, IMG_ORIENTATION, &set_value);
    pi_camera_reg_get(&camera, IMG_ORIENTATION, &reg_value);
    printf("img orientation %d\n",reg_value);

    #ifdef QVGA_MODE
    set_value=1;
    pi_camera_reg_set(&camera, QVGA_WIN_EN, &set_value);
    pi_camera_reg_get(&camera, QVGA_WIN_EN, &reg_value);
    printf("qvga window enabled %d\n",reg_value);
    #endif

    #ifndef ASYNC_CAPTURE
    set_value=0;
    pi_camera_reg_set(&camera, VSYNC_HSYNC_PIXEL_SHIFT_EN, &set_value);
    pi_camera_reg_get(&camera, VSYNC_HSYNC_PIXEL_SHIFT_EN, &reg_value);
    printf("vsync hsync pixel shift enabled %d\n",reg_value);
    #endif
```

Configure camera registers

Hands-on: Image acquisition and filtering

Include drivers

Include image IO library

Include own demosaicking function

Define acquisition size

Define variables – place buffer in L2

Open and initialize camera

Open camera

Configure camera re

Allocated buffers in L2

Set up OS, then jump to test_camera

Hands-on: Image acquisition and filtering

Include drivers

Include image IO library

Include own demosaicking fu

Define acquisition size

Define variables – place buff

Open and initialize camera

```c
#include "pmsis.h"
#include "bsp/bsp.h"
#include "bsp/camera/himax.h"

#define WIDTH    324
#ifdef QVGA_MODE
#define HEIGHT   244
#else
#define HEIGHT   324
#endif

PI_L2 unsigned char *buff;

PI_L2 unsigned char *buff_demosaick;

static struct pi_device camera;
static volatile int done;

static void handle_transfer_end(void *arg)
{
    done = 1;
}

static int open_camera(struct pi_device *d
{
    printf("Opening Himax camera\n");
    struct pi_himax_conf cam_conf;
    pi_himax_conf_init(&cam_conf);

#if defined(QVGA_MODE)
    cam_conf.format = PI_CAMERA_QVGA;
#endif

    pi_open_from_conf(device, &cam_conf);
    if (pi_camera_open(device))
        return -1;
    pi_camera_control(device, PI_CAMERA_CM
```

Asynchronus capture – can queue buffer before starting camera

Start camera

Wait for capture to end (pi_yield() blocks until an event happens)

Blocking capture

Stop and close camera

Apply a kernel

Write image over openOCD/JTAG to a file on the computer

```c
    #ifdef ASYNC_CAPTURE
    // Start up async capture task
    done = 0;
    pi_task_t task;
    pi_camera_capture_async(&camera, buff, BUFF_SIZE, pi_task_callback(&task, handle_transfer_end, NULL));
    #endif

    // Start the camera
    pi_camera_control(&camera, PI_CAMERA_CMD_START, 0);
    #ifdef ASYNC_CAPTURE
    while(!done){pi_yield();}
    #else
    pi_camera_capture(&camera, buff, BUFF_SIZE);
    #endif

    // Stop the camera and immediately close it
    pi_camera_control(&camera, PI_CAMERA_CMD_STOP, 0);
    pi_camera_close(&camera);


    #ifdef COLOR_IMAGE
    demosaicking(buff, buff_demosaick, WIDTH, HEIGHT, 0);
    #else
    demosaicking(buff, buff_demosaick, WIDTH, HEIGHT, 1);
    #endif

    // Write to file
    #ifdef COLOR_IMAGE
    WriteImageToFile("../../../img_color.ppm", WIDTH, HEIGHT, sizeof(uint32_t), buff_demosaick, RGB888_IO);
    #else
    WriteImageToFile("../../../img_gray.ppm", WIDTH, HEIGHT, sizeof(uint8_t), buff_demosaick, GRAY_SCALE_IO);
    #endif

    WriteImageToFile("../../../img_raw.ppm", WIDTH, HEIGHT, sizeof(uint8_t), buff, GRAY_SCALE_IO );

    pmsis_exit(0);
```

Hands-on: Image acquisition and filtering

Include drivers

Include image IO library

Include own demosaicking function

Define ac

Define variables – place buffer in L2

Asynchronus capture callback

Open and initialize camera

But we do not only want to take one image,
we want to continously take images in a loop!
For simplicity, we focus on synchronus capture

Configure camera registers

Allocated buffers in L2

Asynchronus capture – can queue buffer before starting camera

Start camera

Wait for capture to end (pi_yield() blocks until an event happens)

Blocking capture

Stop and close camera

Apply a kernel

Write image over openOCD/JTAG to a file on the computer

Set up OS, then jump to test_camera

Hands-on: Image acquisition and filtering

Include drivers

Include image IO library

Include own demosaicking

Define ac

Define variables – place bu

Open and initialize camera

```
126
127     // Start the camera
128     pi_camera_control(&camera, PI_CAMERA_CMD_START, 0);
```

Start camera

```
132     pi_camera_capture(&camera, buff, BUFF_SIZE);
133     #endif
```

Blocking capture

```
135     // Stop the camera and immediately close it
136     pi_camera_control(&camera, PI_CAMERA_CMD_STOP, 0);
137     pi_camera_close(&camera);
```

Stop and close camera

```
140     #ifdef COLOR_IMAGE
141     demosaicking(buff, buff_demosaick, WIDTH, HEIGHT, 0);
142     #else
143     demosaicking(buff, buff_demosaick, WIDTH, HEIGHT, 1);
144     #endif
```

Apply a kernel

```
146     // Write to file
147     #ifdef COLOR_IMAGE
148     WriteImageToFile("../../../img_color.ppm", WIDTH, HEIGHT, sizeof(uint32_t), buff_demosaick, RGB888_IO);
149     #else
150     WriteImageToFile("../../../img_gray.ppm", WIDTH, HEIGHT, sizeof(uint8_t), buff_demosaick, GRAY_SCALE_IO);
151     #endif
152
153     WriteImageToFile("../../../img_raw.ppm", WIDTH, HEIGHT, sizeof(uint8_t), buff, GRAY_SCALE_IO );
154
155     pmsis_exit(0);
```

Write image over openOCD/JTAG to a file on the computer

Hands-on: Image acquisition and filtering

Include drivers

Include image IO library

Include own demosaicking

Define ac

Define variables – place bu

But we do no
we want to c
For simplicity

Open and initialize camera

```
126
127     // Start the camera
128     pi_camera_control(&camera, PI_CAMERA_CMD_START, 0);
while(1){
132     pi_camera_capture(&camera, buff, BUFF_SIZE);
133     #endif
134
138
139
140     #ifdef COLOR_IMAGE
141     demosaicking(buff, buff_demosaick, WIDTH, HEIGHT, 0);
142     #else
143     demosaicking(buff, buff_demosaick, WIDTH, HEIGHT, 1);
144     #endif
145
146     // Write to file
147     #ifdef COLOR_IMAGE
148     WriteImageToFile("../../../img_color.ppm", WIDTH, HEIGHT, sizeof(uint32_t), buff_demosaick, RGB888_IO);
149     #else
150     WriteImageToFile("../../../img_gray.ppm", WIDTH, HEIGHT, sizeof(uint8_t), buff_demosaick, GRAY_SCALE_IO);
151     #endif
152
153     WriteImageToFile("../../../img_raw.ppm", WIDTH, HEIGHT, sizeof(uint8_t), buff, GRAY_SCALE_IO );
154 }
135     // Stop the camera and immediately close it
136     pi_camera_control(&camera, PI_CAMERA_CMD_STOP, 0);
137     pi_camera_close(&camera);
```

Great! You now have basically an universal pipeline for any kernel you want to run.

Apply a kernel

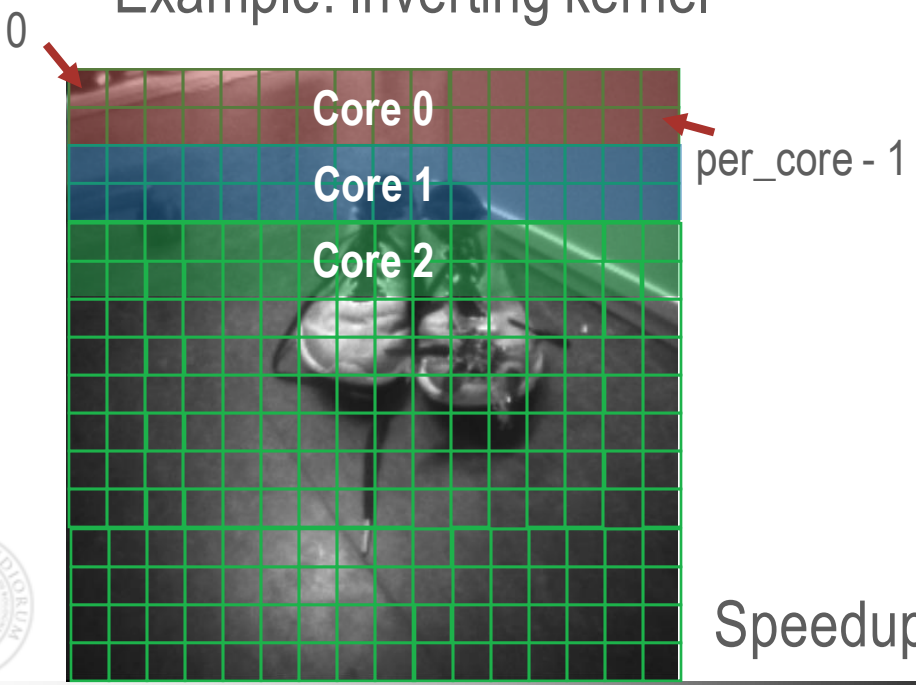
Write image over openOCD/JTAG to a file on the computer

Stop and close camera

Hands-on: Image acquisition and filtering

How do we improve performance?

- Avoid float operations
- Parallelize code
  - All cores should execute similar code on different data
- Example: Inverting kernel




```c
typedef struct {
    char *srcBuffer;     // pointer to the input vector
    char *resBuffer;     // pointer to the output vector
    uint32_t width;      // image width
    uint32_t height;     // image height
    uint32_t nPE;        // number of cores
    uint32_t grayscale;        // grayscale if one
} plp_example_kernel_instance_i32;
```

```c
void cluster_inverting(void* args)
{
    uint32_t idx = 0;
    uint32_t core_id = pi_core_id(), cluster_id = pi_cluster_id();
    plp_example_kernel_instance_i32 *a = (plp_example_kernel_instance_i32*)args;
    char *srcBuffer = a->srcBuffer;
    char *resBuffer = a->resBuffer;
    uint32_t width = a->width;
    uint32_t height = a->height;
    uint32_t nPE = a->nPE;

    uint32_t total = width*height;

    // amount of elements per core, rounded up
    uint32_t per_core = (total+nPE-1)/nPE;
    // compute the last element of the area each core has to process
    uint32_t upper_bound = (core_id+1)*per_core;
    // as we always rounded up before (to distribute the load as equal as possible)
    // we need to check if the upper bound is still in our matrix
    if(upper_bound > total ) upper_bound = total;
    // loop over the area assigned to the core
    for (idx = core_id*per_core; idx < upper_bound; idx++) {

            resBuffer[idx] = 255 - srcBuffer[idx];

    }
}
```

Speedup: @50MHz FC and Cluster from 8ms ->1.5ms