PLATFORM FOR EFFICIENT DEVELOPMENT OF IMAGE ACQUISITION AND PROCESSING SYSTEMS

Peter Reichel, Optical Sensors Group, Fraunhofer IIS / EAS, Dresden
Overview

- Introduction
- Components and Architecture
- Vision task design
- Summary
Overview

- Introduction
- Components and Architecture
- Vision task design
- Summary
Introduction
Classical Machine Vision

- camera & PC used for observation and control
- two separate tasks: Image acquisition and image processing
  - acquisition, output and transfer of image data
    → Images can be seen as a common “language”
  - all components (camera, PC, software) exchangeable separately
- **But**: Data transfer bottleneck between camera and PC!
Introduction

Vision Chips

- image acquisition and processing: **No images leave the sensor!**
  - image processing and feature extraction as early as possible
  - advantage: low bandwidth requirements sensor ↔ (post-)processing

- programmable Vision Chips: Image-Sensor-Processor
  - Processing Elements (PE) can be used for different Vision Tasks
  - flexible and modular tool-chain required
Introduction

Vision Chips

- image acquisition and processing: **No images leave the sensor!**
  - image processing and feature extraction as early as possible
  - advantage: low bandwidth requirements sensor ↔ (post-)processing

- programmable Vision Chips: Image-Sensor-Processor
  - Processing Elements (PE) can be used for different Vision Tasks
  - **flexible and modular tool-chain required**
Overview

- Introduction
- Components and Architecture
- Vision task design
- Summary
Platform's architecture

Basic idea

- closing the gap between on-chip processing and concrete Vision Tasks
- three different abstraction levels
  - Lowest (c): ASIP based control system [Reichel15a]
  - Middle (b): adjustable assembler
  - Highest (a): Python based programming environment
Platform's architecture

Adjustable assembler

- supports multiple control flows
- dependencies between different ASIPs
- easily adjustable by libraries

```plaintext
1 .import "asipl"
2 .import "asip2"
3
4 [main1:asipl] {
5   ...
6   add A,B
7   print A
8   sync_notify "asip2"
9   ...
10 }
11
12 [main2:asip2] {
13   ...
14   sync_wait "asip1"
15   ...
16 }
17
18 [tab1:asipl]
19 table <? math.sqrt(25) ?>table for asip1
```

Source: [Reichel16]
Platform's architecture

Python integration

- **Python**: platform independence, many different libraries, ...

- individual functions or macros can be assigned to a specific ASIP
  - function: subset of Python bytecode directly translated into ASM
  - macro: called from functions and executed directly

- ... or run regularly on the host to
  - control the translation process and
  - parametrize the VSoC and post process the received data

Source: [Reichel16]
Overview

- Introduction
- Components and Architecture
- Vision task design
- Summary
Vision task design
Programmable Skeletons

- **library** containing VSoC basic image processing functions
- **Skeletons** for the abstraction of image acquisition / -processing
  - selection of basic skeleton by requirements
  - adaption by registration of functions in pre-defined slots

Source: [Reichel16]
Vision task design

Holistic design

- **Holistic design** – integration of
  - **pre-processing on VSoC** and
  - **post-Processing on Host** (e.g. by using OpenCV)

```python
1 # instance of the vsoc connection
2 vsoc = VSoC(...)  # (a)
3
4 # own processing function
5 @simd.assign()
6 def my_proc():  # (b)
7    ...
8
9 # skeleton instance
10 skel = skeletons.SimpleImgRead(vsoc)  # (c)
11 skel.reg_slot("ADC", adc.single_slope)
12 skel.reg_slot("PROC", my_proc)
13 ...
14
15 # conversion and programming
16 vsoc.program(convert_and_asm(...))  # (d)
17
18 # set region-of-interest
19 skel.set("roi", (0,0),(1024,1024))  # (e)
20
21 # image acquisition and processing
22 import cv2 as cv
23 import numpy as np
24 krl = np.ones((5,5),np.uint8)
25
26 while True:  # (f)
27    img=skel.acquire_img()
28    erosion = cv.erode(img,krl)
29    cv.imshow("Image", erosion)
30 ...```

Overview

- Introduction
- Components and Architecture
- Vision task design
- Summary
Conclusion

- Vision Chips: Image processing and feature extraction as early as possible!
  - **But:** Gap between on-chip processing and concrete Vision Tasks
- Python based programming environment
  - library elements with basic image processing functions
  - programmable Skeletons for the abstraction of image acquisition / –processing
- holistic design of vision tasks
  - image acquisition and –processing on VSoC and host
  - incorporation of available libraries – OpenCV, NumPy, …
Thank you!

Literature:

- [Reichel15b] Reichel et al., “Simulation Environment for a Vision-System-on-Chip with Integrated Processing”, International Conference on Distributed Smart Cameras (ICDSC), 2015