

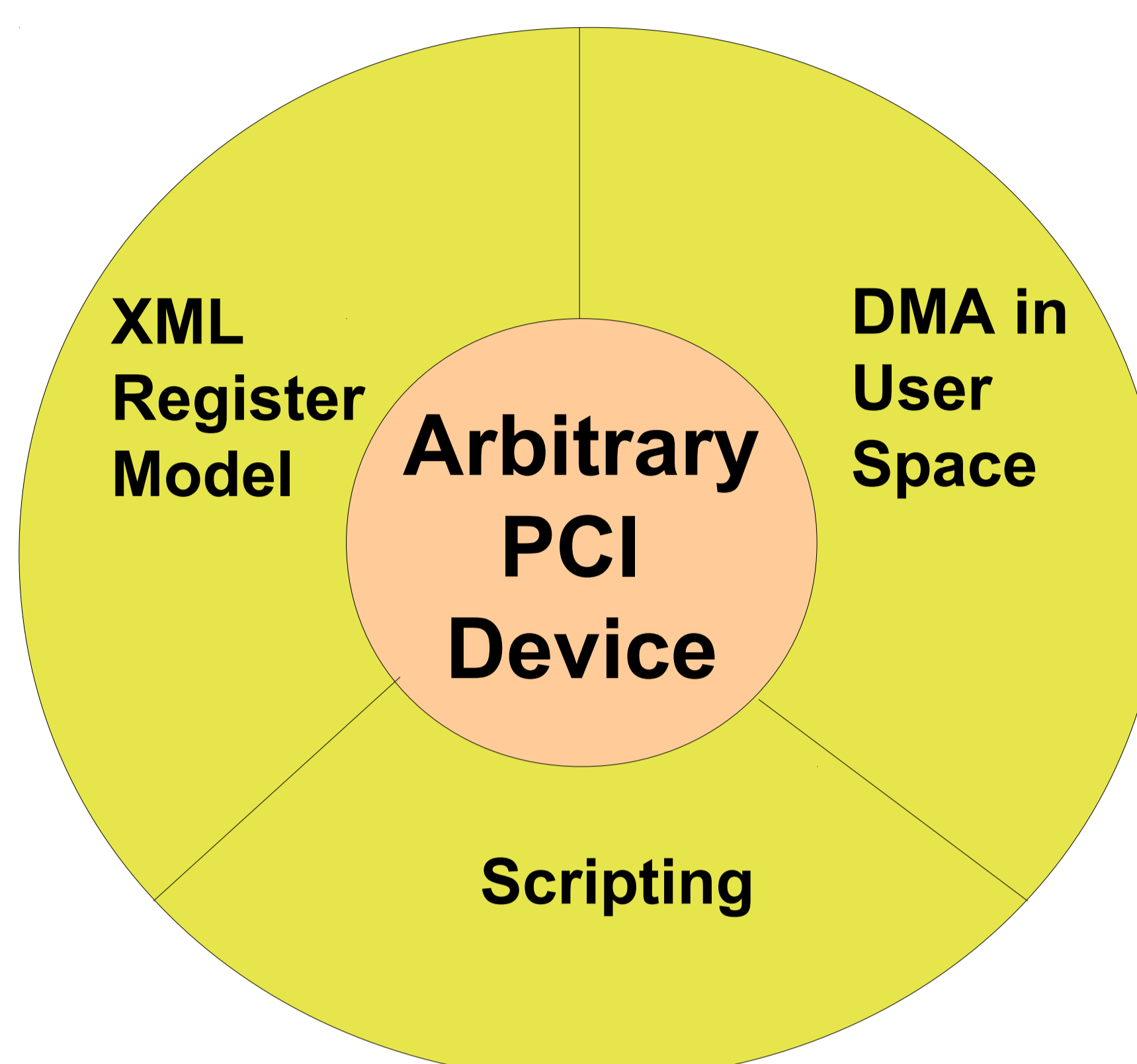
S. Chiligaryan, M. Caselle, A. Kopmann, U. Stevanovic, M. Vogelgesang

Karlsruhe Institute of Technology, Karlsruhe, Germany

ALPS (Advanced Linux PCI Services) are a flexible toolset to prototype and debug new PCI-based DAQ hardware using an universal driver.

Motivation

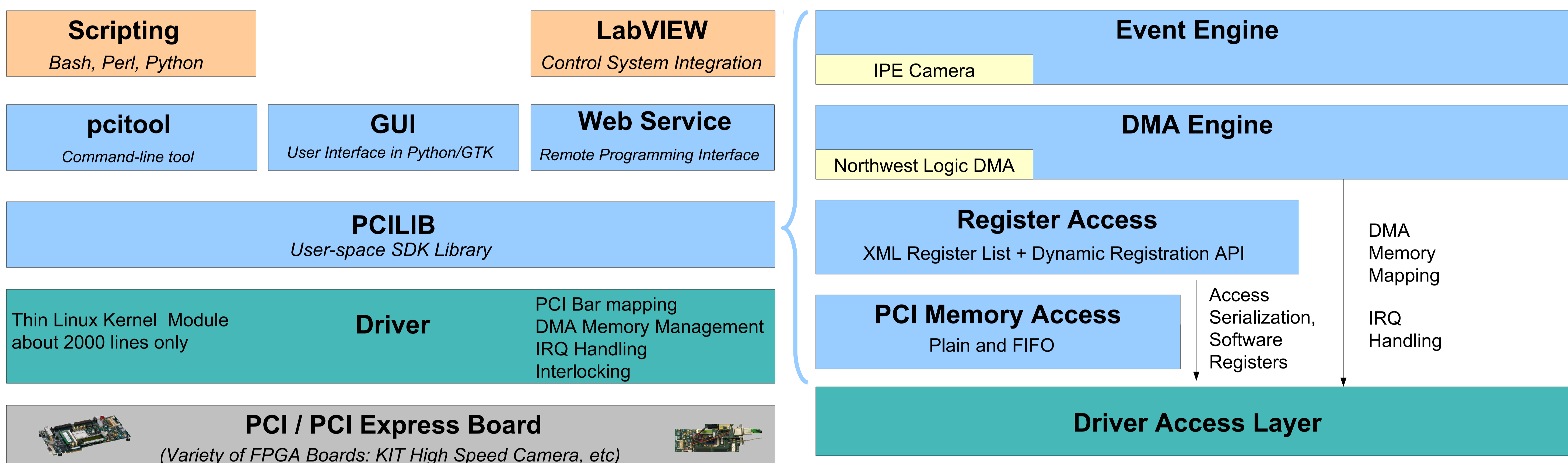
Writing stable and performant drivers and keeping them up to date with the latest Linux kernel is a complex and tedious task. It is especially difficult to synchronize parallel development of hardware and software. However, many components of PCI driver are standard. Basically, in development phase hardware engineers often only need access to the device registers and the ability transfer data between device and host memory in few different modes. This functionality may be provided uniformly for most devices by a universal driver. So, the hardware design is not blocked by missing or malfunction software and no software modifications are required for hardware debugging.



Features

- Tiny and easy to support kernel module
- XML-based register model
- Access by address or name
- 8-64 bit little/Big-endian access
- Support of bit-fields
- Data Transfers
 - Plain
 - FIFO register
 - High-speed DMA support
- Register/DMA scripting support
- Device specific functions using plugins
- Web service API (planned)
- Binding to multiple languages

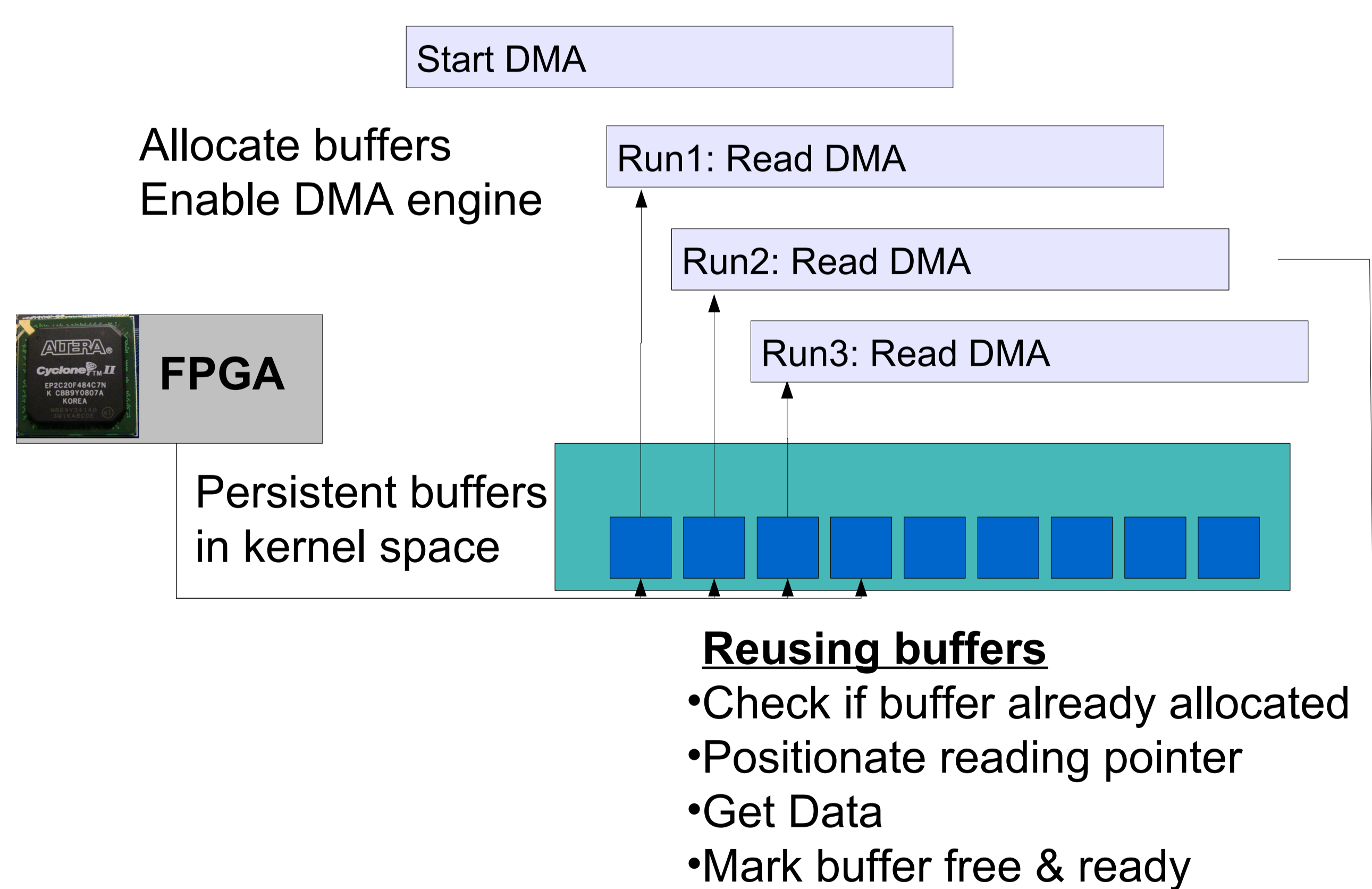
Architecture



ALPS consists of the tiny kernel driver, SDK library (pcilib), and command-line tool (pcitool). The GUI and Web Service interface are planned.

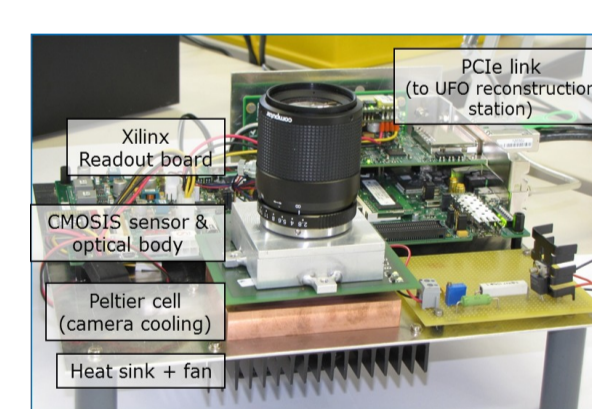
There are 4 layers in SDK library: raw access to the PCI I/O memory, register model, DMA engine, and device specific code.

DMA Engine in User Space



- DMA implemented in user-space
- Tiny and easy to support kernel module responsible for synchronization and memory management
- Easily extensible to new DMA protocols without kernel-level programming
- Persistent kernel buffers
- Scripting and debugging support
- Read/Write/Peek functionality
- Page/Package/Buffer access levels
- High performance
- 1350 Mb/s camera is tested with real-time frame decoding

Example



KIT High Speed Streaming Camera
Resolution: 2048 x 1088 @ 10 bits:
Frame Rate: 300 fps
Data Rate: 1350 MB/s

```
pci --start-dma dma1
pci -g -o images.raw --run-time 60000000 &
pid=$!
for i in `seq 1 100000`; do
  old_size=`ls -la images.raw | cut -d " " -f 5`
  pci --trigger
  new_size=`ls -la images.raw | cut -d " " -f 5`
  usleep 100000
  if [ $old_size -eq $new_size ]; then
    echo "Incomplete frame..."
    killall -SIGINT pci
    break
  fi
done
wait $pid
pci --stop-dma
```

```
pci -r status
status = 0x80080000

pci -r 0x9000 -s 16 -a 32
9000: 00007300 00000000 00007300 00000000
9010: 000b7300 00000000 000b7300 00000000
9020: 00000004 00000000 00000004 00000000
9030: 00000004 00000000 00000004 00000000

pci --list-dma-buffers
Buffer  Status  Total Size
-----
0      U FL  4 KB
747    U FL  4 KB
748    U F  4 KB
749    U L  8 B
750    0 B

U - Used, E - Error, F - First block, L - Last Block
```

Example of script used for debugging of the high-speed camera to find a problem with camera trigger signals (left). The presented script initializes DMA engine and starts grabbing frames in the background process. The software triggers are sent in the loop. If after trigger is sent, the size of file does not change within 100 milliseconds, the grabbing thread is killed and the script is terminated without stopping DMA engine. Then, hardware developer can investigate the status registers, state of DMA engine, etc (right). It is possible to see that the last DMA message has extra 8 bytes which could be the source of problem.

