PCIe, MIPI CSI-2, HDMI and DDR memory across platforms from an Open Source perspective

Alexander Wirthmüller
aw@mpsitechnologies.com

Introduction

About me

- Based in Munich
- Diploma in Electrical Engineering

- Founder and Director at MPSI Technologies
- MPSI Technologies: make Embedded Software development more fun by replacing repetitive tasks by model-based source code generation

- (Freelance) roles as R&D engineer for cutting-edge projects requiring FPGA-based subsystems – one at a time

The new normal of low-cost FPGA interfaces

- Former justification to use FPGA's: multiple SPI/UART/GPIO – now covered by modern MCU's
- Major FPGA use case still is high-bandwidth custom data processing
- Requires high-bandwidth data sources and sinks as well as storage (memory)

Why …

… PCI Express?
↳ talk to (PC) host
… MIPI CSI-2?
↳ any video processing can use a camera input
… HDMI (out)?
↳ quick visualization of video processing results even with (Linux) host present
… DDR memory?
↳ internal BlockRAM too small for megapixel images in mid-range devices

Why not …

… Ethernet *MII?
↳ in most cases useful only with protocols on top (e.g. UDP/TCP). This is better done from within an OS like Linux
… HDMI (in)?
↳ beyond scope, postponed
… USB 3.0?
↳ non-typical FPGA interface, postponed
… JESD204*?
↳ beyond scope, RF ADC/DAC almost always comes along with very expensive hardware

Mission statement

For this presentation and for the underlying effort

- All mid-range devices since 2010 support at least some of the “new normal low-cost” FPGA interfaces
- BUT: low-cost doesn’t imply low-effort (making “low-cost” relative)
- AND: these interfaces are one major obstacle for switching between vendors easily

- Collect and publish information on interfaces at a public location on the web, keep it up-to-date
- Offer Open Source wrappers which make interface access vendor-neutral
- Identify Open Source software and vendor-provided examples which can be re-used for other vendors’ platforms
- Implement a fully documented and easy-to-reproduce reference application on all suitable platforms
- Do not look at advanced features – rather: block data transfer with PCIe, standard image streaming with MIPI, HD monitor output with HDMI, block read/write from/to DDR memory
- Focus on affordable mid-range devices, limit custom hardware to adapter PCB’s

Mid-range FPGA landscape *)

Vendor	Product line	Year (node)	LE (range)	CPU	PCIe	MIPI	HDMI 1)	DDR
AMD (Xilinx)	Artix 7	2011 (28nm)	13k-215k	-	yes	ext. pass.	yes	PHY muxed
	Zynq	2011 (28nm)	23k-444k	ARMv7	some	ext. pass.	yes	PHY dedic. 2)
	Zynq UltraScale+ CG	2015 (16nm)	81k-600k	ARMv8	some	I/O std.	yes	PHY dedic. 2)
Efinix	Titanium	2021 (16nm)	36k-176k 3)	soft RISC-V	no 3)	hard IP	yes	PHY dedic.
intelPSG (Altera)	Cyclone V SE	2012 (28nm)	25k-110k	ARMv7	no	ext. pass.	yes	PHY dedic. 2)
	Cyclone V GX	2011 (28nm)	36k-301k	-	yes	ext. pass.	yes	PHY muxed
	Agilex 5	2023 ("7"nm)	50k-656k	ARMv8	Yes	TBD	yes	TBD
Lattice	CrossLink-NX	2020 (28nm)	17k-39k	-	some	hard IP	yes	I/O feat.
Microchip (-semi)	PolarFire SoC	2019 (28nm)	23k-461k	RISC-V	yes	I/O std.	yes	PHY dedic. 2)

1) HDMI only requires 165MHz LVDS for full HD, 2) shared with CPU complex, some with additional PHY muxed,
3) larger capacities and PCIe-capable SerDes announced

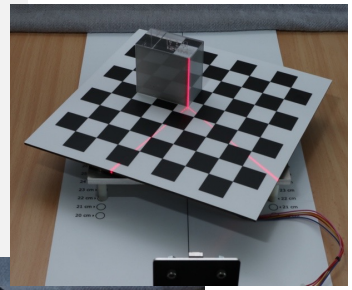
*) non-exhaustive, “cost optimized” in some vendors’ slang
Information from vendor sources retrieved online in June 2023

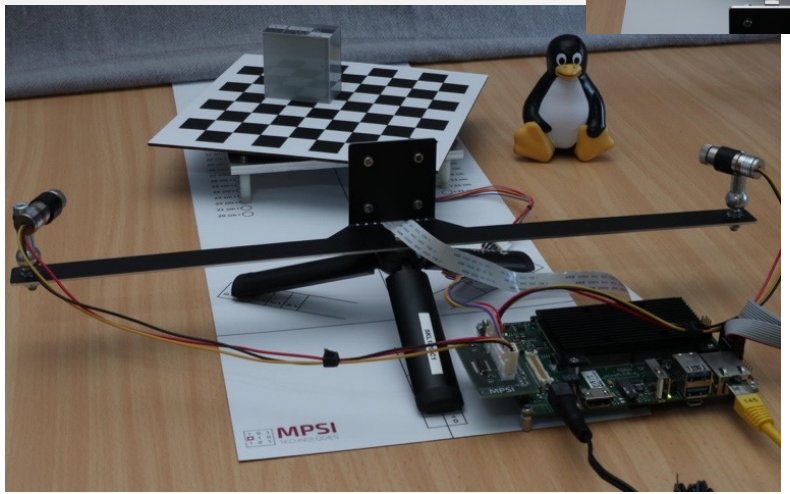
A typical mid-range project

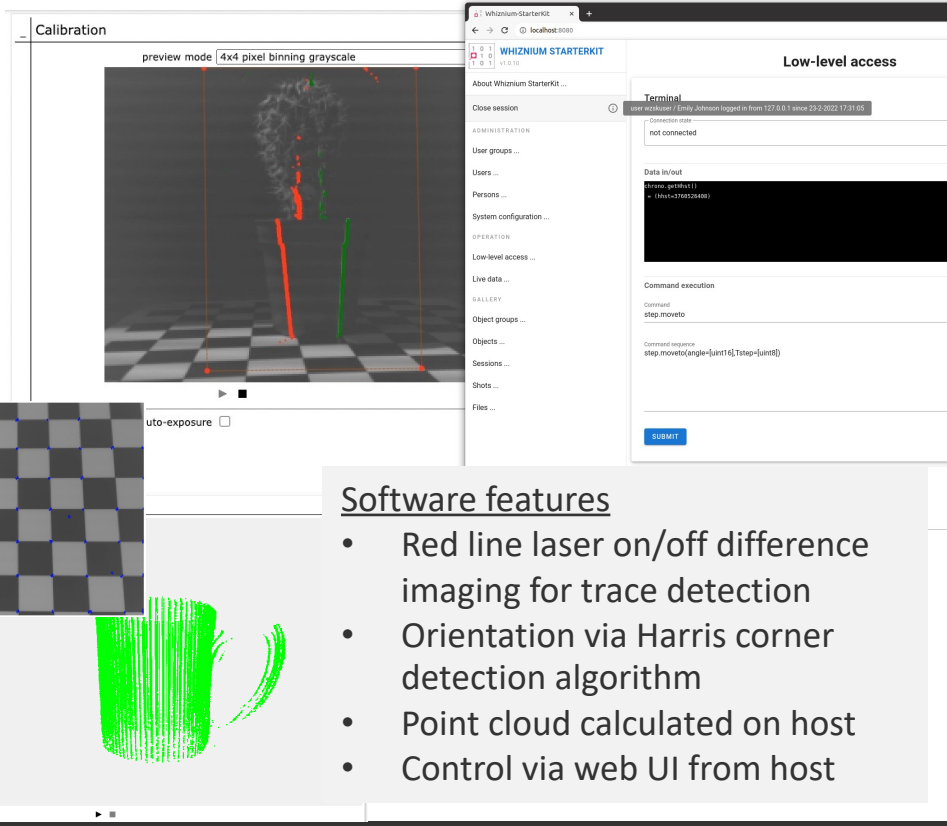
Original version

Hardware features

- Turntable with stepper motor
- Tripod with camera/laser holder
- OV5640 five megapixel camera with parallel interface
- Two adjustable red line lasers





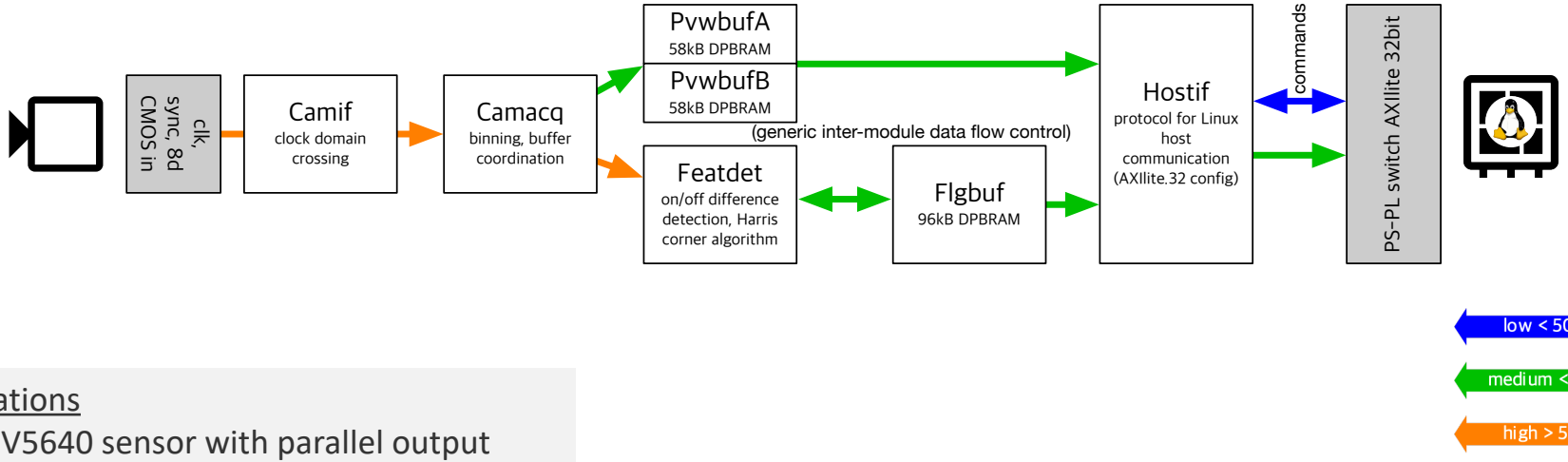


Software features

- Red line laser on/off difference imaging for trace detection
- Orientation via Harris corner detection algorithm
- Point cloud calculated on host
- Control via web UI from host

A typical mid-range project

Original AMD Zynq 7010 implementation

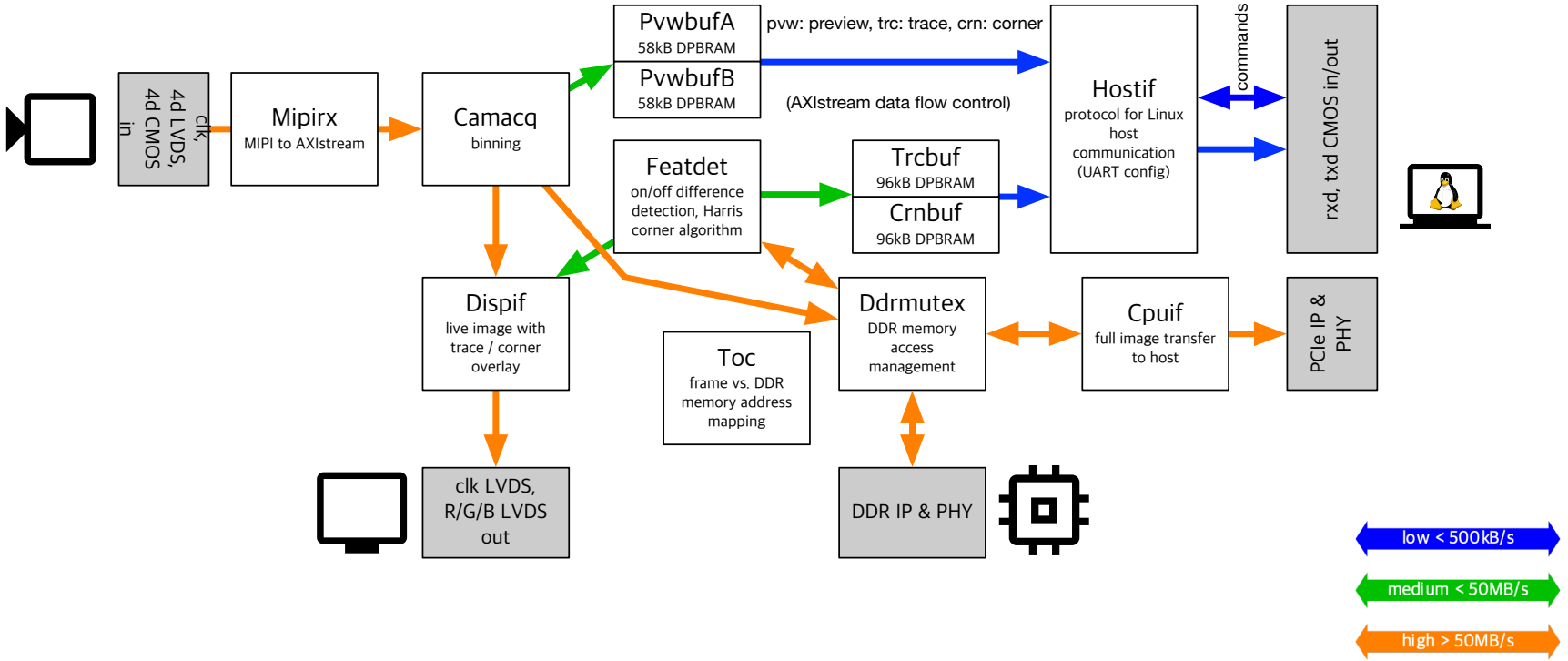


Limitations

- OV5640 sensor with parallel output only on market (and obsolete)
- On-FPGA storage allows for one bit per pixel at full resolution
- Not even possible to copy whole frames to host

A typical mid-range project

Adding extras / Digilent Nexys Video with AMD Artix 7 implementation

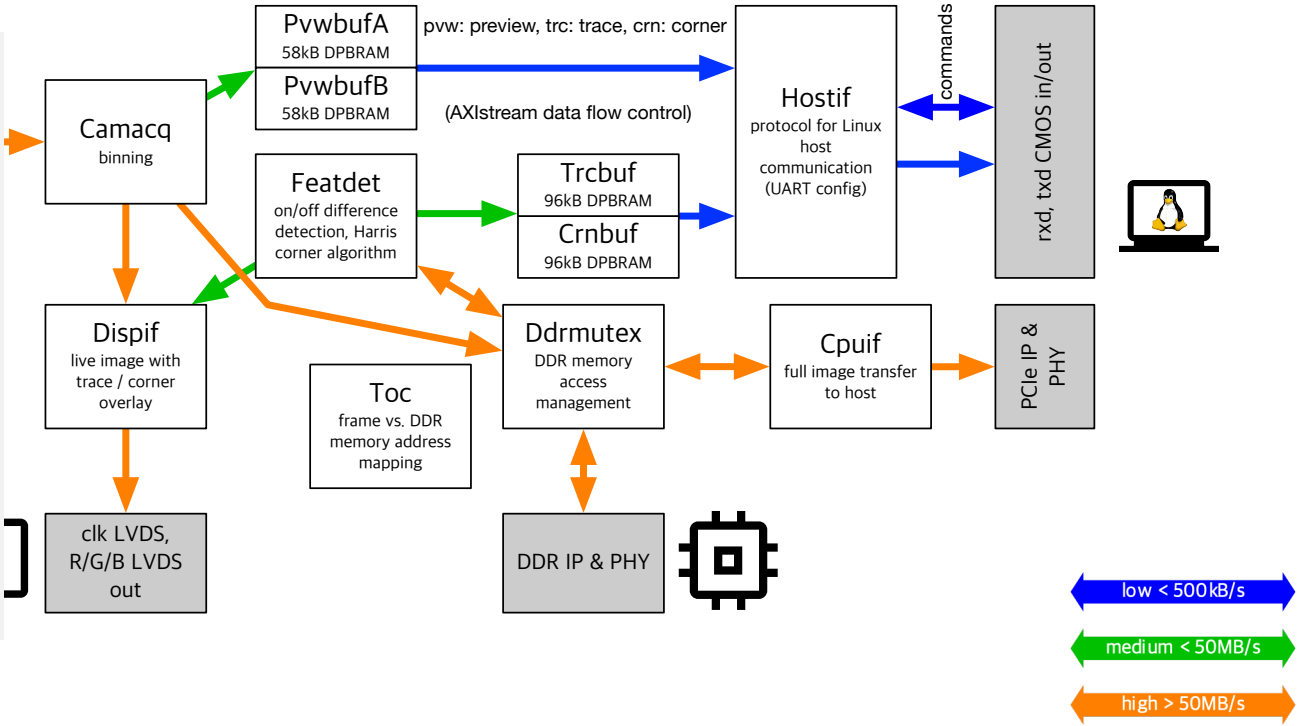


A typical mid-range project

Adding extras / Digilent Nexys Video with AMD Artix 7 implementation

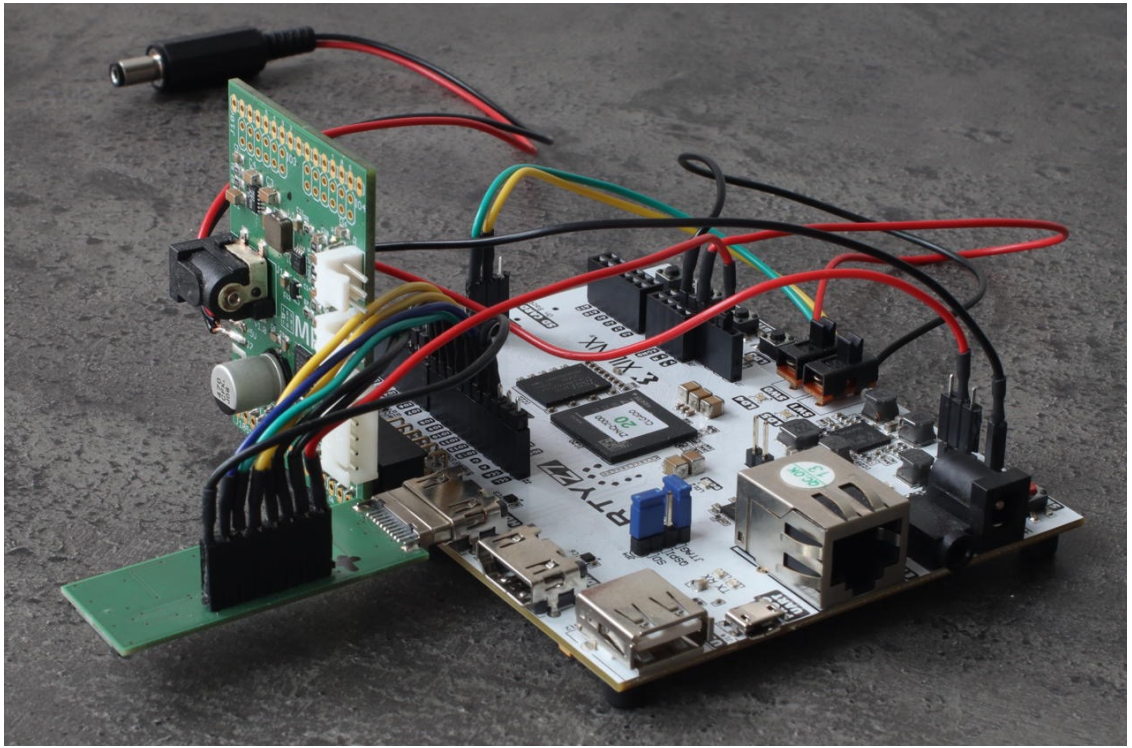
Improvements

- Compatible with many MIPI cameras offering multiple resolution choices
- DDR memory allows HDR image capture (multi exposure times to counter laser saturation)
- DDR memory plus PCIe allows transferring whole frames to host
- HDMI output allows for quick verification without host



FPGA(-SoC) project variants

AMD (Xilinx)



Digilent Arty Z7-20 (arty)

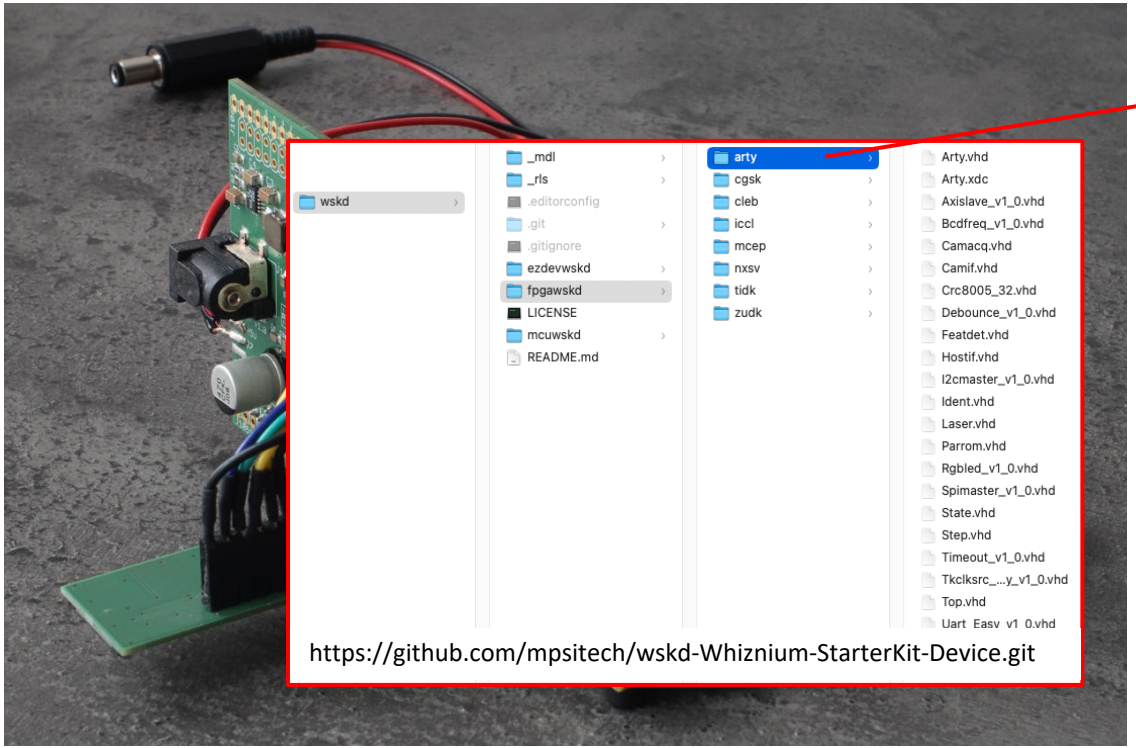
- Based on Zynq Z7020
- $299 (digilent.com)
- Device XC7Z020-1CLG400C
- €128 (mouser.de)

- Host: Yocto Linux on ARMv7

✗ PCIe
✓ MIPI CSI-2 via HDMI in
✓ HDMI out
✓ 512MB DDR3 SDRAM

FPGA(-SoC) project variants

AMD (Xilinx)



Digilent Arty Z7-20 (arty)

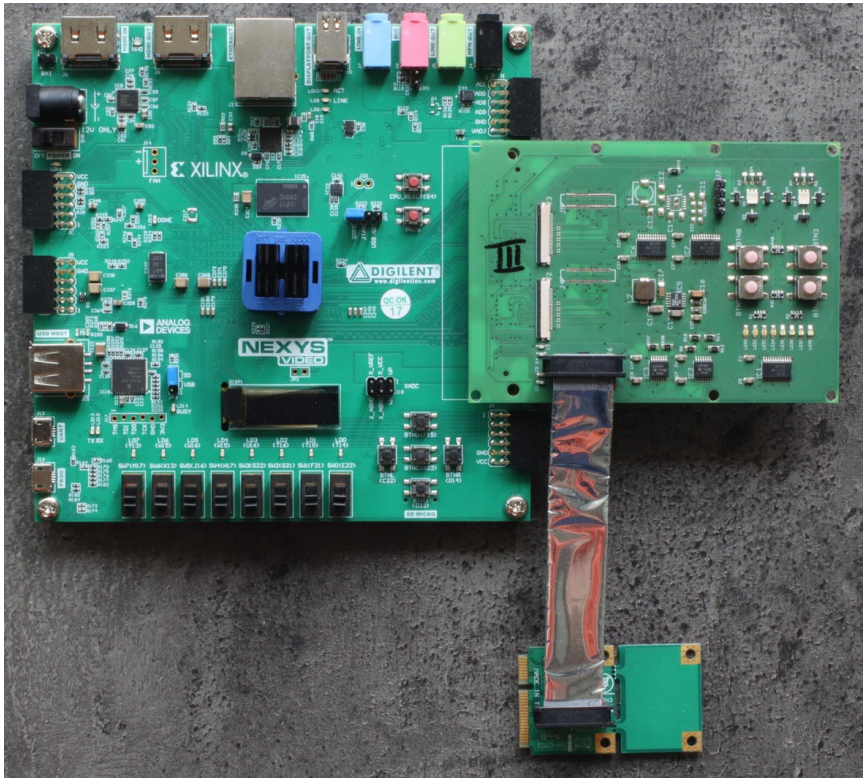
- Based on Zynq Z7020
- $299 (digilent.com)
- Device XC7Z020-1CLG400C
- €128 (mouser.de)

- Host: Yocto Linux on ARMv7

✗ PCIe
✓ MIPI CSI-2 via HDMI in
✓ HDMI out
✓ 512MB DDR3 SDRAM

FPGA(-SoC) project variants

AMD (Xilinx)



Digilent Nexys Video (nxsv)

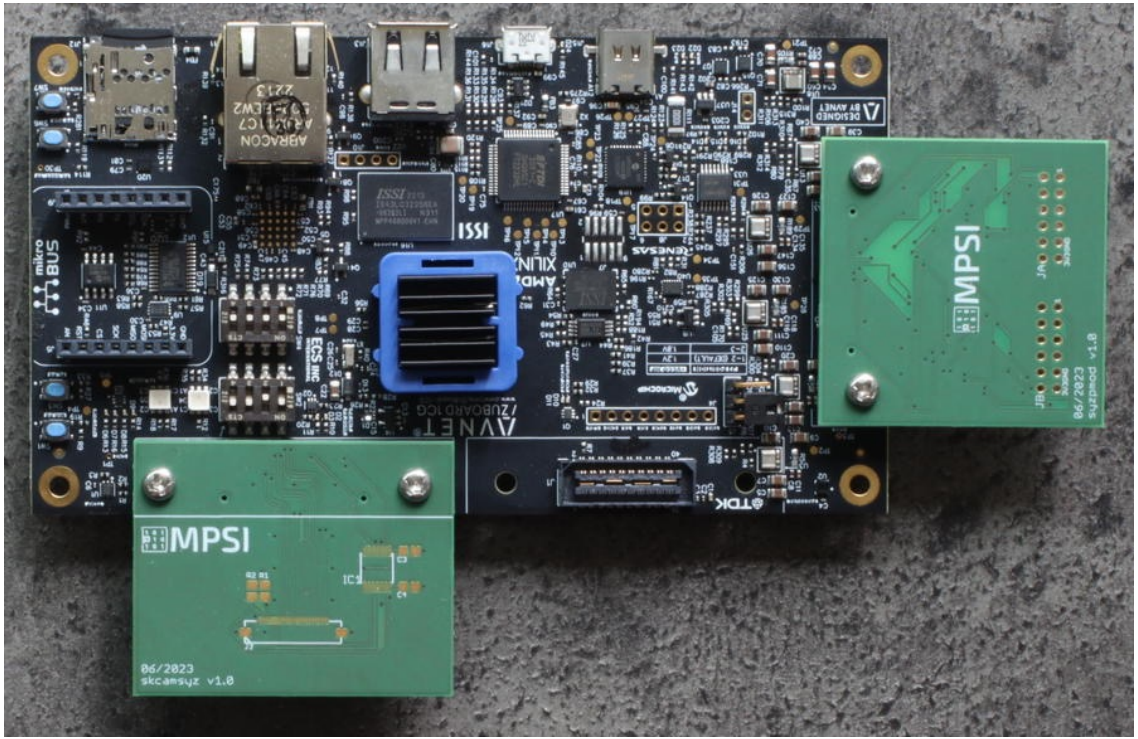
- Based on Artix 7
- $549 (digilent.com)
- Device XC7A200T-1SBG484C
- €277 (mouser.de)

- Host: PC / NUC

- ✓ PCIe via FMC
- ✓ MIPI CSI-2 via FMC
- ✓ HDMI out
- ✓ 512MB DDR3 SDRAM

FPGA(-SoC) project variants

AMD (Xilinx)



Avnet ZUBoard 1CG (zudk)

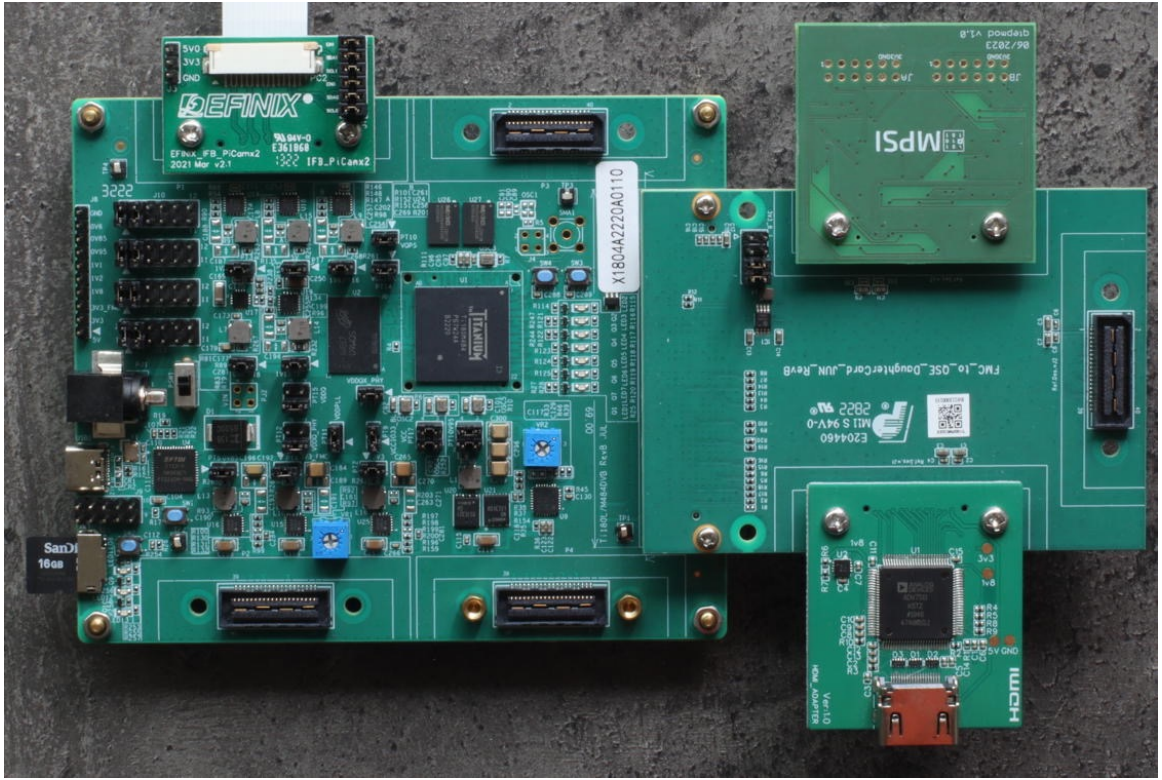
- Based on Zynq UltraScale+ 1CG
- €149 (de.farnell.com)
- Device ZU1CG-1SBVA484I
- €290 (!) (mouser.de)

- Host: Yocto Linux on ARMv8

✗ PCIe
✓ MIPI CSI-2 via SYZYGY TXR
✗ HDMI out
✓ 1GB DDR4 SDRAM

FPGA(-SoC) project variants

Efinix



Efinix Titanium Ti180 dev kit (tidk)

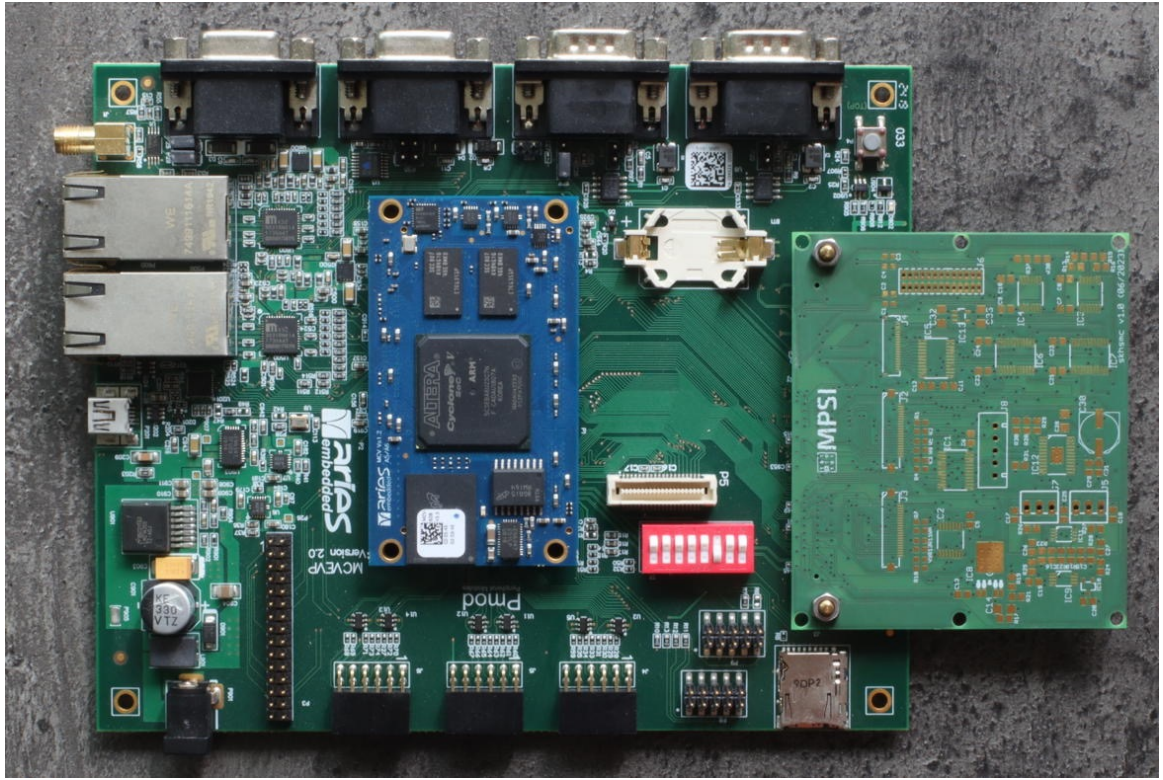
- €776 (digikey.de)
- Device Ti180M484I3
- Price N/F

- Host: Buildroot Linux on soft RISC-V

✗ PCIe
✓ MIPI CSI-2 via SYZYGY
✓ HDMI out via FMC + SYZYGY(-like)
✓ 32MB LPDDR4 SDRAM

FPGA(-SoC) project variants

intelPSG (Altera)



Aries Cyclone V SE SoM (mcep)

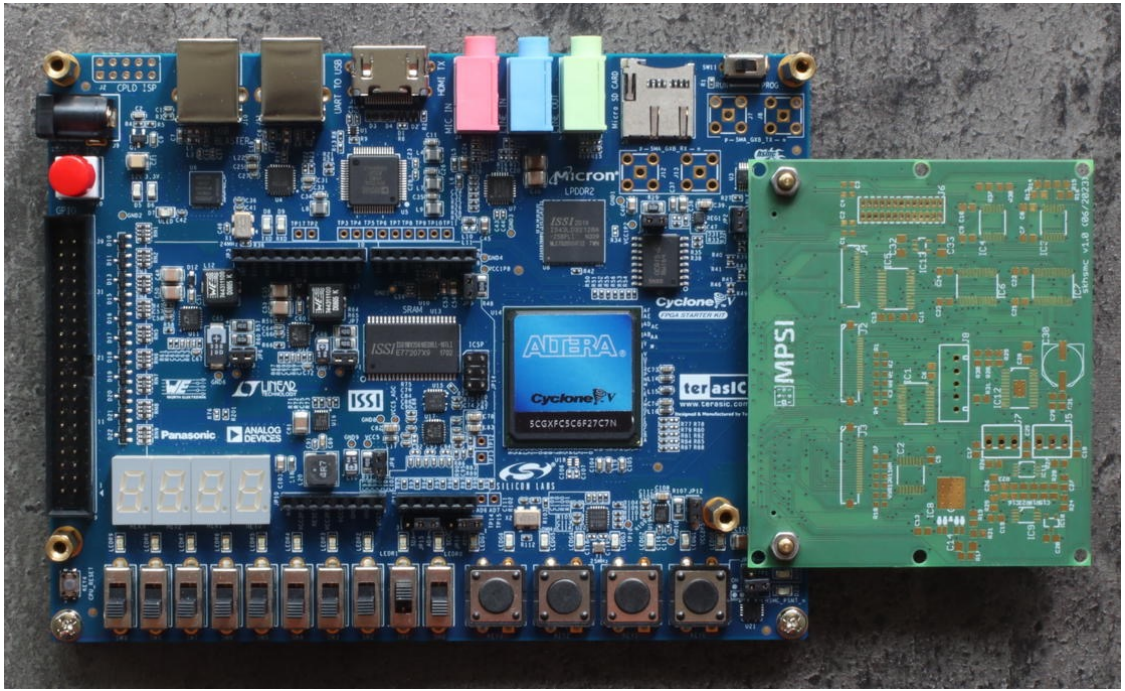
- MCV-6DB SoM on EVP (base board)
- €309 SoM only (aries-embedded.de)
- Device 5CSEBA6U23C7N
- €272 (!) (digikey.de)

- Host: Yocto Linux on ARMv7

✗ PCIe
✓ MIPI CSI-2 via HSMC
✗ HDMI out
✓ 1GB DDR3 SDRAM

FPGA(-SoC) project variants

intelPSG (Altera)



terasIC Cyclone V GX st. kit (cgsk)

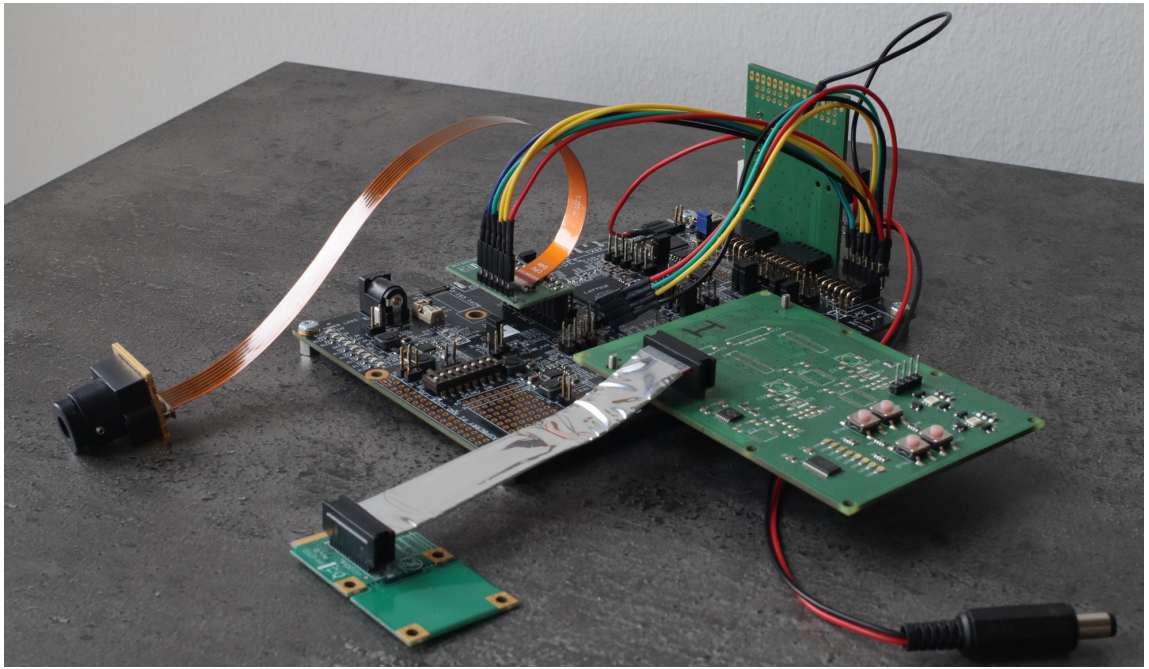
- €259 (digikey.de)
- Device 5CGXFC5C6F27C7N
- €227 (!) (digikey.de)

- Host: PC / NUC

✓ PCIe via HSMC
✓ MIPI CSI-2 via HSMC
✓ HDMI out
✓ 512MB LPDDR2 SDRAM

FPGA(-SoC) project variants

Lattice



Lattice CrossLink-NX eval. board (cleb)

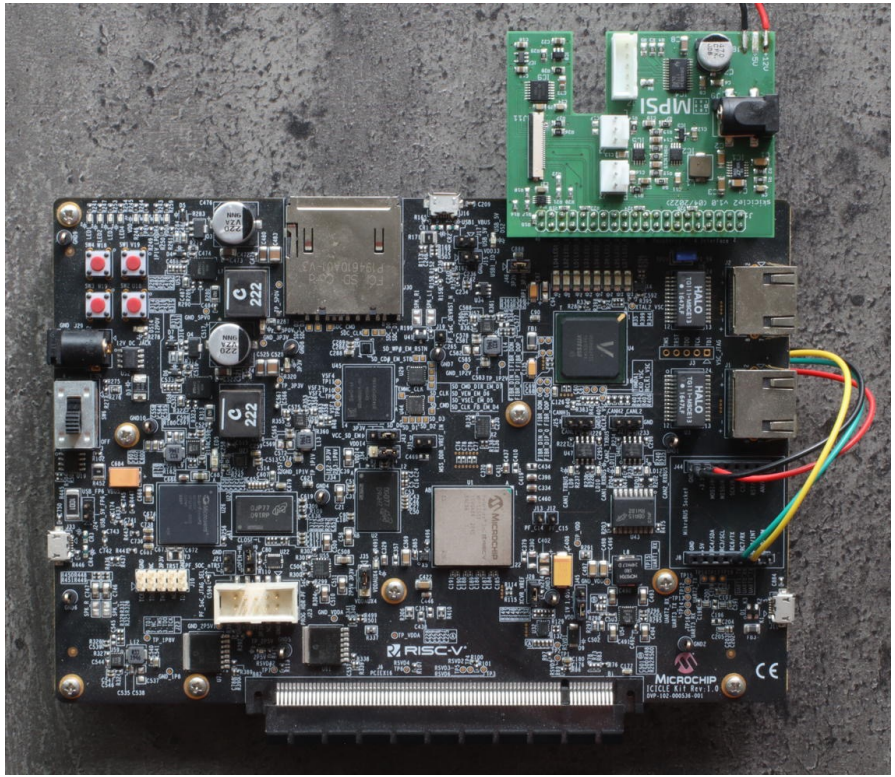
- €118 (digikey.de)
- Device LIFCL-40-9BG400C
- €64 (digikey.de)

- Host: PC / NUC

✓ PCIe via FMC
✓ MIPI CSI-2
✗ HDMI out
✗ DDR memory

FPGA(-SoC) project variants

Microchip (-semi)



Microchip PolarFire SoC Icicle kit (iccl)

- €549 (digikey.de)
- Device MPFS250T-FCVG484EES
- Price N/F

- Host: Yocto Linux on RISC-V

✓ (PCIe)
✓ (MIPI CSI-2)
✗ HDMI out
✓ 2GB LPDDR4 SDRAM

Resources

- Up-to-date project documentation on GitHub (to receive all information of this presentation and more)

  https://mpsitech.github.io/Laser-Scanner-By-Platform

- Documentation of Whiznium, Open Source tool for model-based source code generation covering the “embedded full-stack” (gives instructions to set up the tools themselves but also SoC-related things like Yocto/Buildroot)

  https://github.com/mpsitech/The-Whiznium-Documentation

- The Whiznium Starter Kit source code (Linux daemon)

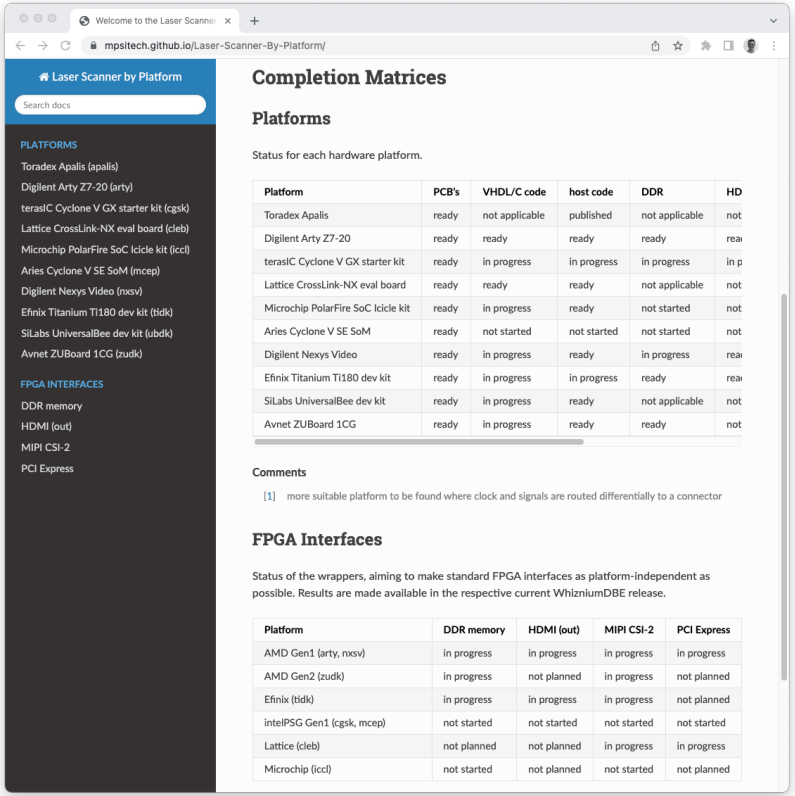
  https://github.com/mpsitech/wzsk-Whiznium-StarterKit

- The Whiznium Starter Kit Device source code (RTL for all platforms)

  https://github.com/mpsitech/wskd-Whiznium-StarterKit-Device

FPGA(-SoC) project variants

- Expected completion by mid-September 2023
- Weekly updates
- Advice to investigate once “code” columns show “published” for a given platform



PCI Express

Overview | Hardware prerequisites | FPGA integration | Resources

- PCI Express **Gen1** (introduced 2003 as replacement for parallel PCI) offers 2.5 Gbps per lane
- Focus on **x1**, i.e. single data lane, net data rate of 250MB/s

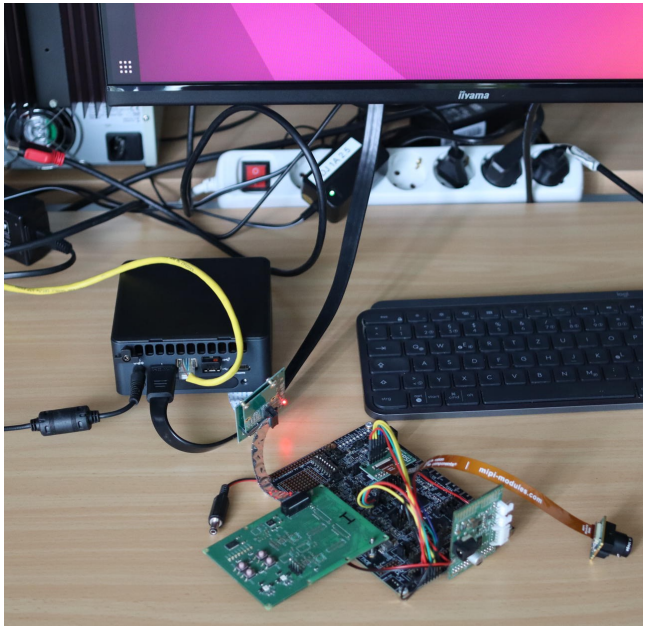
Physical layer:

- Point-to-point communication (here: host-to-FPGA), FPGA is **endpoint**
- NRZ line code: clock (1.25GHz, 0.4ns eye) is recovered from signal
- Reference clock (100MHz) signal only present during system start-up
- Among those presented here, the most demanding interface, impossible without dedicated silicon
- I^2C interface not relevant for FPGA implementation

Configuration / transaction layer:

- Configuration (”capabilities”) read by host from FPGA during system start-up
- Focus on **“Memory Write”** / **”Memory Read”** only with 1k payload size per transaction

- Max. allowed length of PCIe v1.0 lane is 38 cm (15 inch)
- For practical purposes ubuntu Linux host with Mini-PCIe breakout beats workstation / riser card: example intel NUC 11 i3 with converter M.2 Key A+E to mini PCIe slot (DeLOCK 62848) and a mini PCIe breakout (DeLOCK 41370)
- Some Embedded Linux hosts (e.g. Toradex Apalis) also feature mini PCIe slots but this adds complication (Yocto) to an already complicated topic

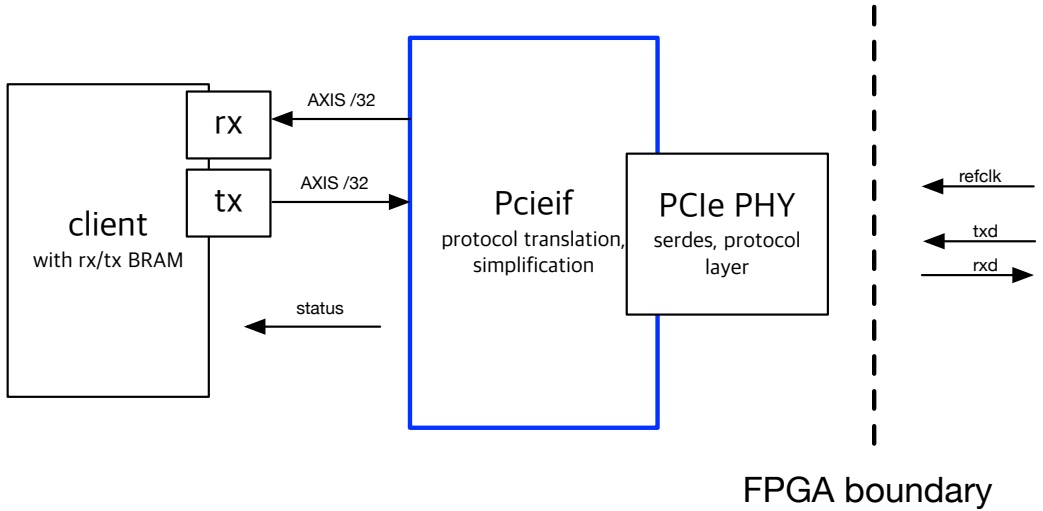


Nice-to-have block structure

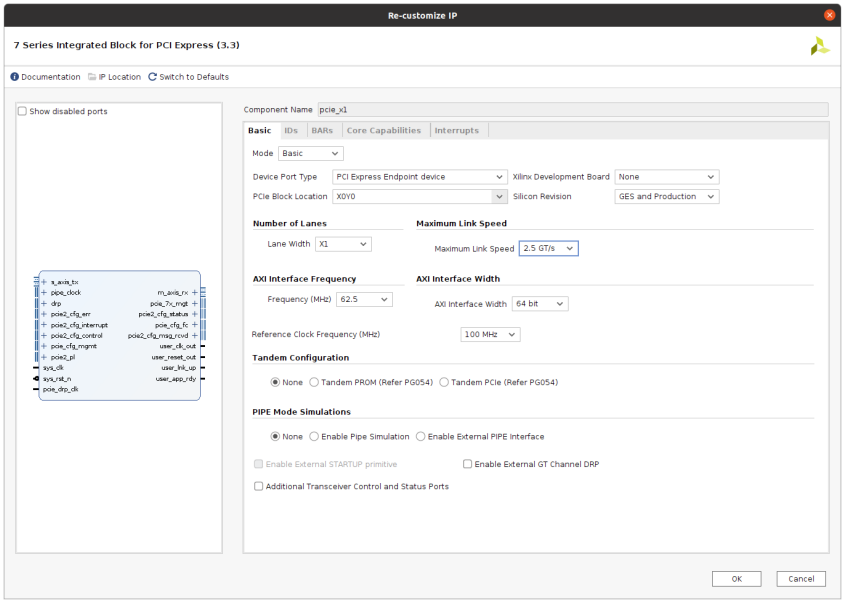
Connection to hardware:
- Reference clock (diff.)
- RXD (diff.)
- TXD (diff.)

Connection to RTL:
- AXI stream data out
- AXI stream data in
  (both: 62.5MHz @ /32, 31.25MHz @ /64)

- Register-based configuration/status

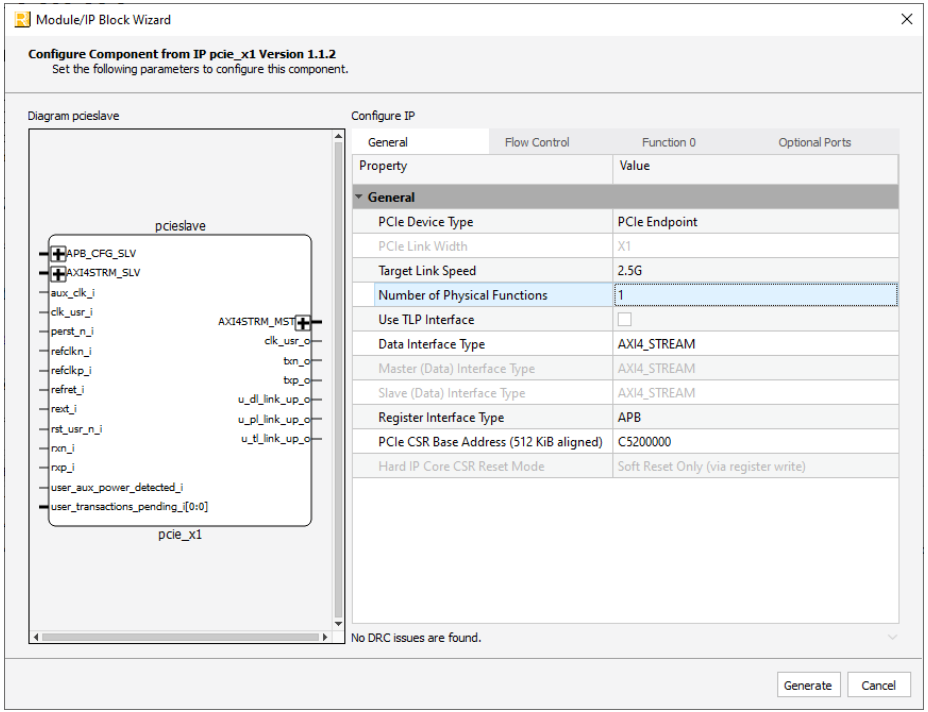


- AMD: AXIS master for RX, AXIS slave tor TX (as seen from FPGA)



- intelPSG: 32 bit Avalon interface

- Lattice: AXIS master for RX, AXIS slave tor TX, APB config with 32 bit address mapped registers



- Linux host perspective: lsusb

```
mpsitech@chip:~$ sudo lspci -v
00:00.0 Host bridge: Intel Corporation Device 9a04 (rev 01)
            DeviceName: Onboard - Other
            Subsystem: Intel Corporation Device 3002
            Flags: bus master, fast devsel, latency 0
            Capabilities: [e0] Vendor Specific Information: Len=14 <?>
...
00:02.0 VGA compatible controller: Intel Corporation Device 9a78 (rev 01) (prog-if 00 [VGA controller])
...
01:00.0 Serial controller: MosChip Semiconductor Technology Ltd. 4-Port PCIe Serial Adapter (prog-if 02 [16550])
            Subsystem: Asix Electronics Corporation (Wrong ID) 4-Port PCIe Serial Adapter
            Flags: fast devsel, IRQ 16
            I/O ports at 3030 [size=8]
            Memory at 4fa07000 (32-bit, non-prefetchable) [size=4K]
            Memory at 4fa06000 (32-bit, non-prefetchable) [size=4K]
...
```

- PCIe typically only makes sense combined with DMA and interrupts
- the Linux kernel's Userspace I/O has corresponding features, already integrated with PCIe

A. Vendor-provided IP documentation, user guides, product guides

1. AMD: 7 Series FPGAs Integrated Block for PCI Express (UG477) [IP encrypted, included in free license]
2. intelPSG: Cyclone V Hard IP for PCI Express (UG-01110) [IP encrypted, included in free license]
3. Lattice: PCIe X1 IP Core - Lattice Radiant Software (IPUG-02091) [IP encrypted, not free (4-hour expiry)]
4. Microchip: PolarFire SoC FPGA PCI Express (UG0920)
   PolarFire FPGA and PolarFire SoC FPGA PCI Express (DS50003188B)

B. Publicly available use cases / examples

C. Relevant work by Open Source community

1. https://github.com/alexforencich/verilog-pcie

MIPI CSI-2

Overview | Hardware prerequisites | FPGA integration | Resources

- MIPI CSI-2 is the (only remaining?) standard for megapixel camera sensors
- Market dominance of SONY IMX range, from ArduCam's to internal iPhone cameras

Physical layer:

- Up to four lanes together with one clock lane, matching via FPGA-internal delay line
- Time-division between "low-power" (LP) 1.2V CMOS and "high-speed" (HS) LVDS modes
- DDR protocol with e.g. 1600 Mbps per lane @800 MHz

Raspberry Pi Cam lineup

Two-lane:

- V2: 8MP IMX219

Optionally four-lane:

- HQ: 12.3MP IMX477
- V3: 11.9MP IMX708

Protocol:

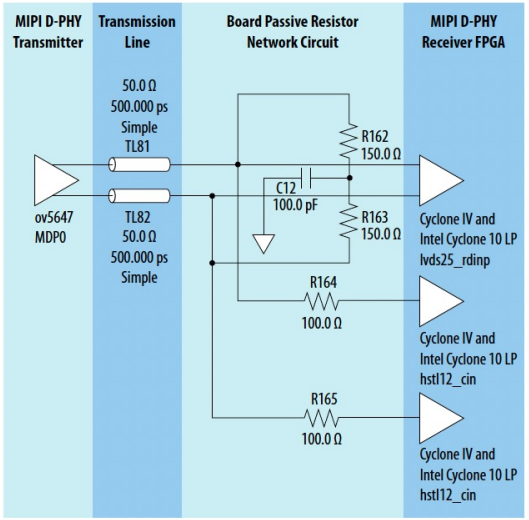
- "LP announces what comes in next HS chunk", bit packing and distribution across lanes (RAW/YUV/…) specified by supplier

Configuration:

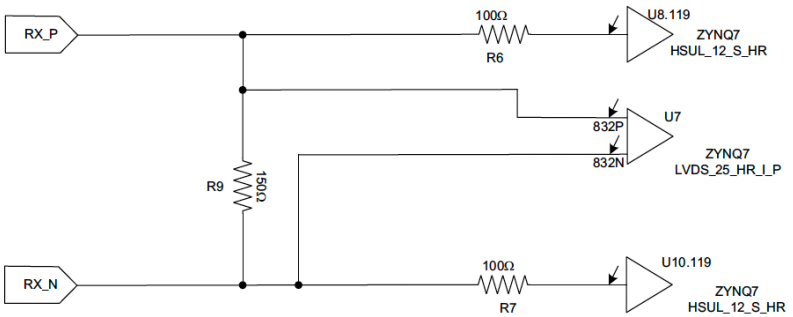
- I^2C interface configures everything: number of lanes, resolution, image format, frame rate, exposure time, …
- without access to datasheet mostly stuck with default (workarounds: Linux kernel sources, contact SONY / FRAMOS)

MIPI CSI-2

- Current generation FPGA(-SoC)'s feature native MIPI support
- AMD and intelPSG each published an application note showing how to split LP / HS externally with passives



(source: intelPSG AN754)



(source: Xilinx XAPP894)

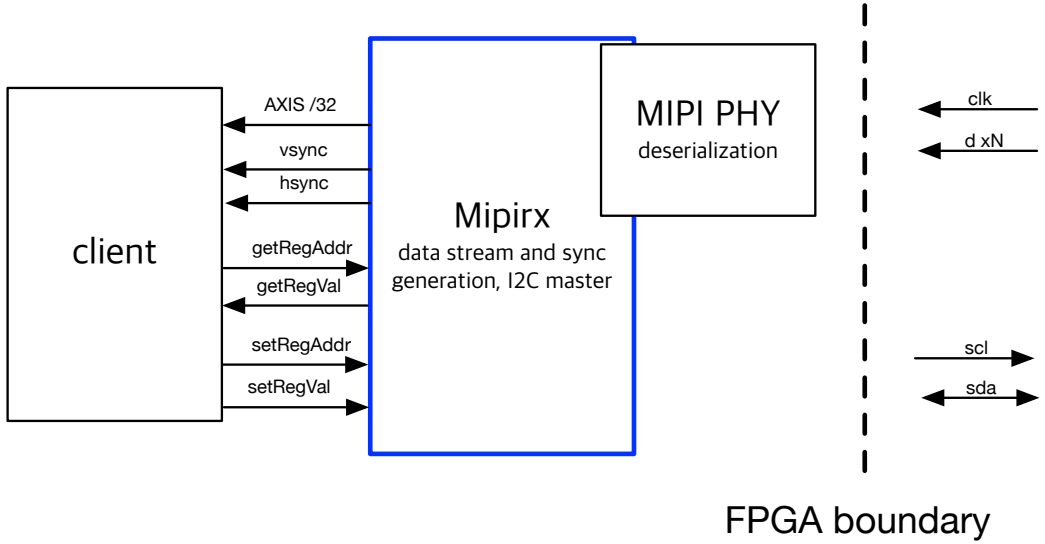
- … but this is recommended for < 800 Mbps only, which is exceeded even by most ArduCam's

Nice-to-have block structure

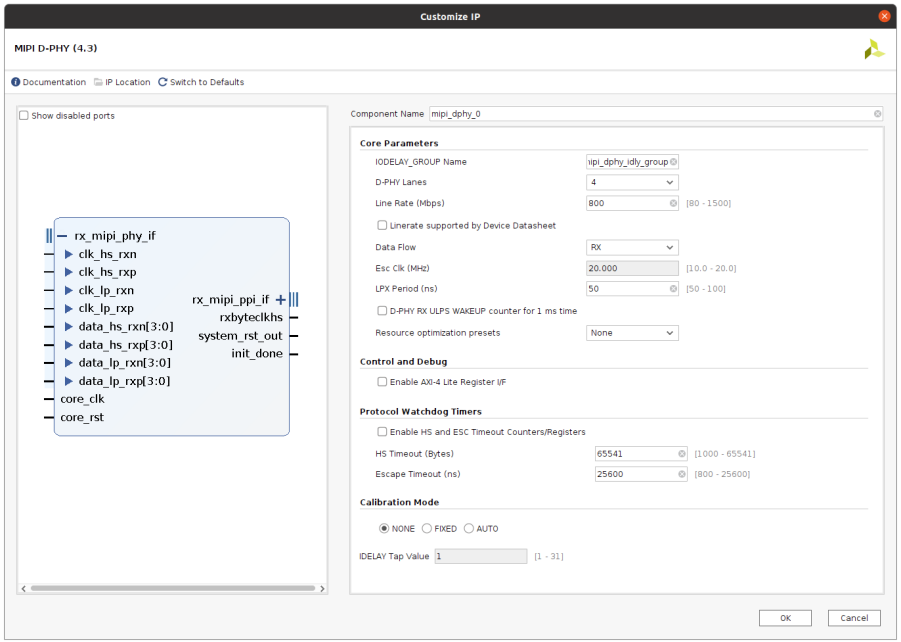
Connection to hardware:
- clock (diff.)
- N lane RX (diff.)
- I^2C

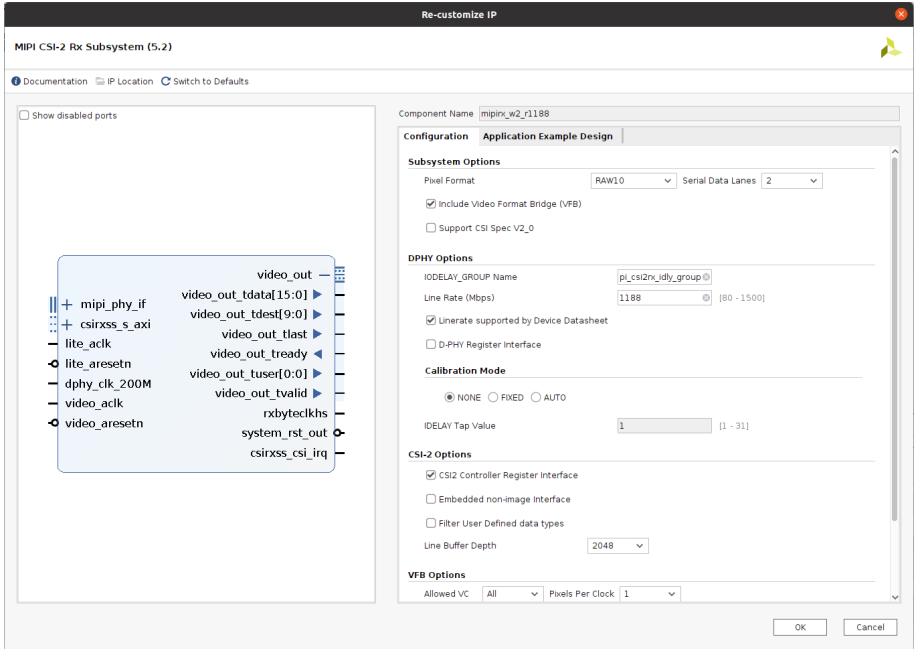
Connection to RTL:
- AXI stream RGB/YUV data out
  (e.g. 5 MP x 10 bit x 60 fps = 3000 Mbps
  => 93.75 MHz @ /32, < 50 MHz @ /64)

- {get/set}RegVal configuration

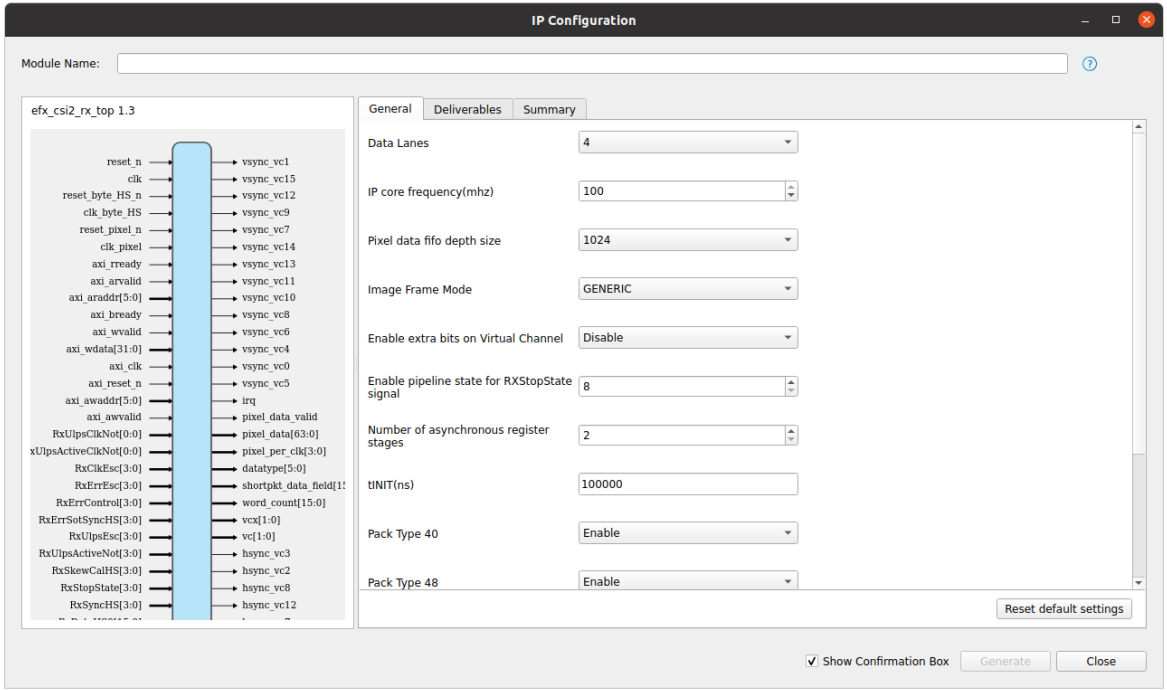


- AMD: per-lane 1:8 gearing vs. AXIS master fixed 16 bit

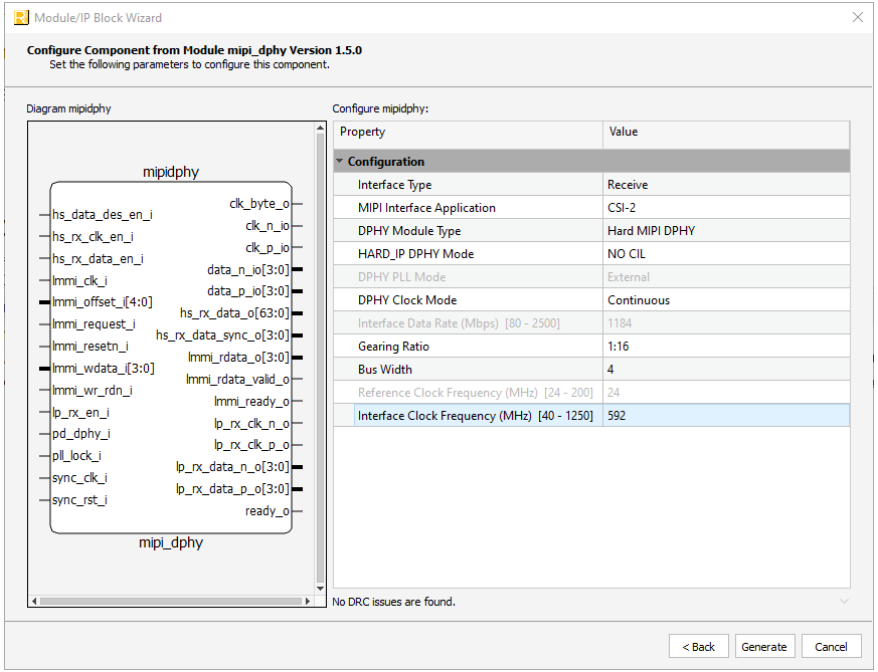


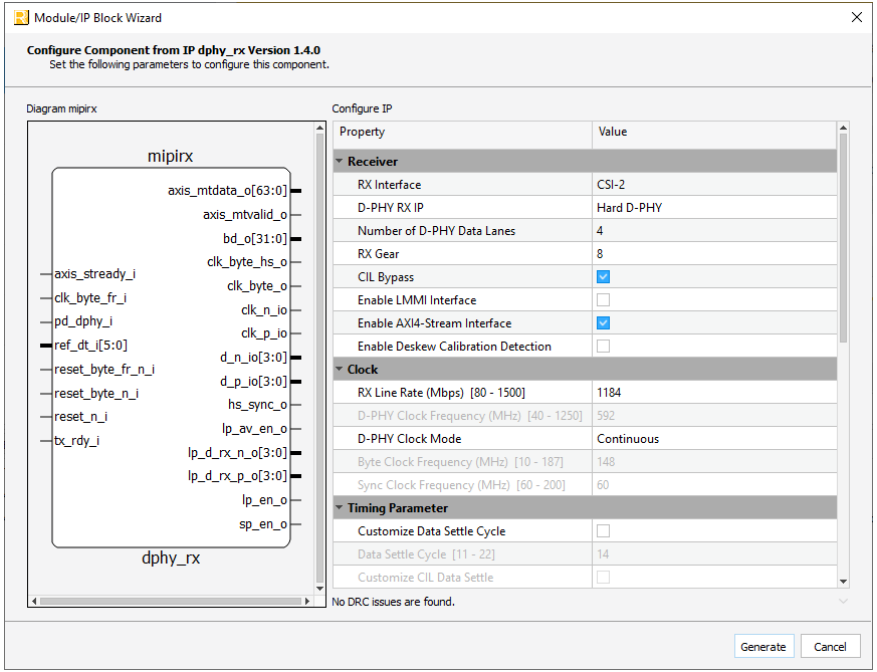


- Efinix: full decoding to 64 bit pixel data plus sync



- Lattice: 4:64 gearing vs. AXIS master





A. Vendor-provided IP documentation, user guides, product guides

1. AMD: 7 Series FPGAs SelectIO Resources (UG471) [included in free license]
   MIPI CSI-2 Receiver Subsystem (PG232) [included in free license]
2. Efinix: Ti180 Data Sheet (DSTi180) chapter on MIPI D-PHY [IP encrypted, included in free license]
3. intelPSG: to be explored
4. Lattice: MIPI D-PHY Module – Lattice Radiant Software (IPUG-2061) [IP encrypted, included in free license]
   CSI-2/DSI D-PHY Rx IP Core - Lattice Radiant Software (IPUG-02081) [IP encrypted, not free (4-hour expiry)]
5. Microchip: PolarFire FPGA and PolarFire SoC FPGA User I/O (DS60001727C)
   MIPI CSI-2 Receiver Decoder For PolarFire (UG0806)

B. Publicly available use cases / examples

C. Relevant work by Open Source community

1. https://github.com/gatecat/CSI2Rx
2. https://github.com/hellgate202/csi2_rx (careful, GPLv3)

HDMI (out)

- HDMI is one of the major standards to transmit image data to screens / projectors

Physical layer:

- Three data lanes (typ. red/green/blue) together with one clock lane running at a 10^{th} of the data speed
- Clock range 25 MHz to 165 MHz (full-HD @ 60Hz)
- TMDS packs 8 bits into 10 bits, i.e. @ 165 MHz, each lane carries 1.65 * 8/10 Gbps = 1.32 Gbps

Protocol:

- vsync and hsync are embedded into the “blue” TMDS lane

HDMI (out)

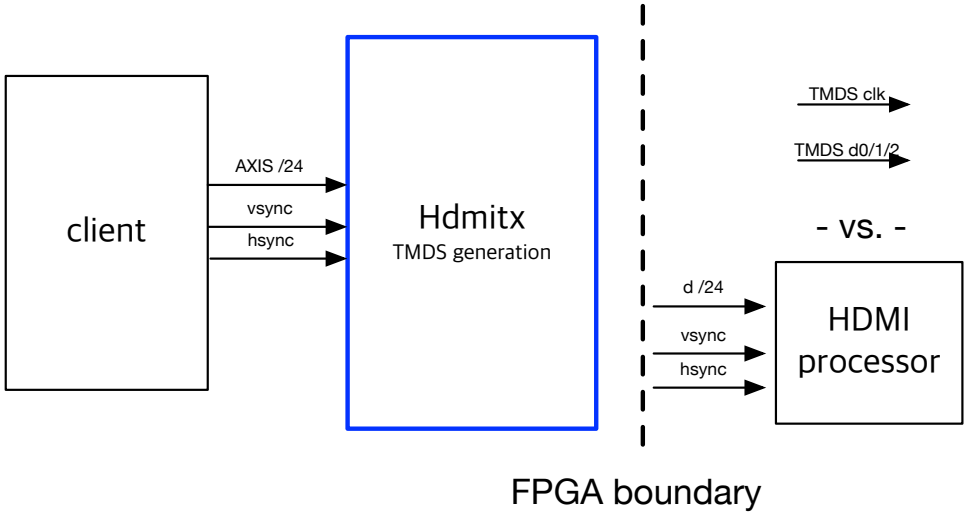
- LVDS transmitters can drive TMDS lines directly (probably not meeting power levels of official spec)
- Standalone (preferred Digilent arty / nxsv): amplifier e.g. TMDS141
- External processor (preferred Efinix tidk / terASIC cgsk): parallel-to-TMDS converter e.g. ADV7511

Nice-to-have block structure

Connection to hardware (standalone):
- TMDS clock (diff.)
- 3x TMDS data (diff.)

Connection to hardware (external processor):
- clock
- 3x 8 .. 12 RGB data
- vsync, hsync

Connection to RTL:
- AXI stream 3x 8 .. 12 RGB data
- vsync, hsync



HDMI (out)

A. Vendor-provided IP documentation, user guides, product guides

B. Publicly available use cases / examples

1. https://www.fpga4fun.com/HDMI.html, https://www.youtube.com/watch?v=sMOZxOCfkBU
2. https://digilent.com/reference/learn/programmable-logic/tutorials/arty-z7-hdmi-demo/start
3. Efinix_Ti180M484_oob out-of-the-box design (requires login)

C. Relevant work by Open Source community

DDR Memory

Overview | FPGA integration | Resources

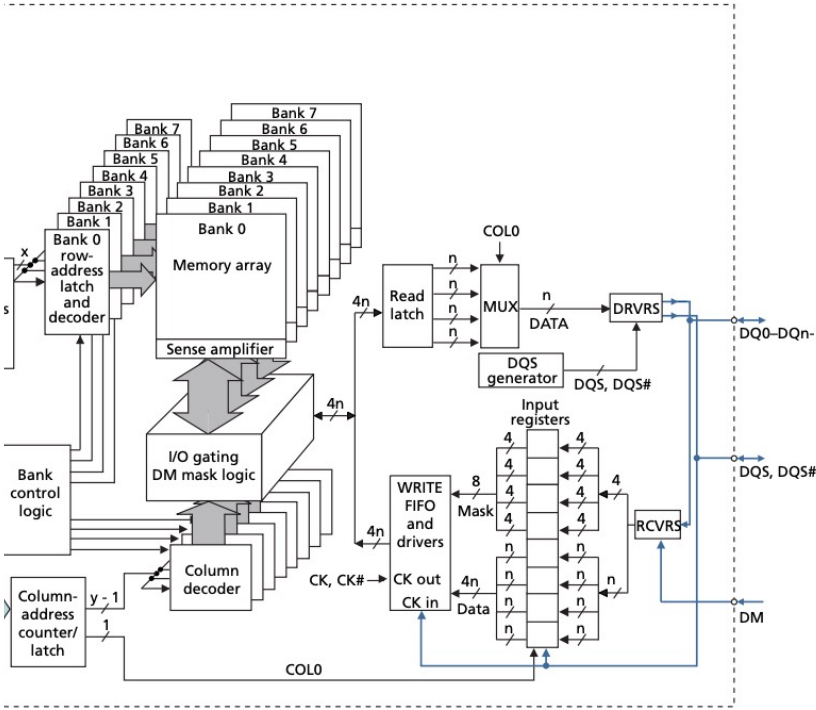
- (LP)DDR{2-6} is the interfacing standard for large dynamic RAM IC's

Physical layer:

- Column / row address inputs (start address)
- Single-ended bidirectional data lanes (typ. 8/16/32 bits) with DDR sampling
- One differential pair of synchronization signals per byte
- DDR protocol e.g. DDR3-800 @400MHz

Protocol:

- Sophisticated "training" procedure in PHY to equalize wire lengths
- Then burst transfer with IC-specific delay from column / row address input to data



(source: MT42L16M32D1 data sheet)

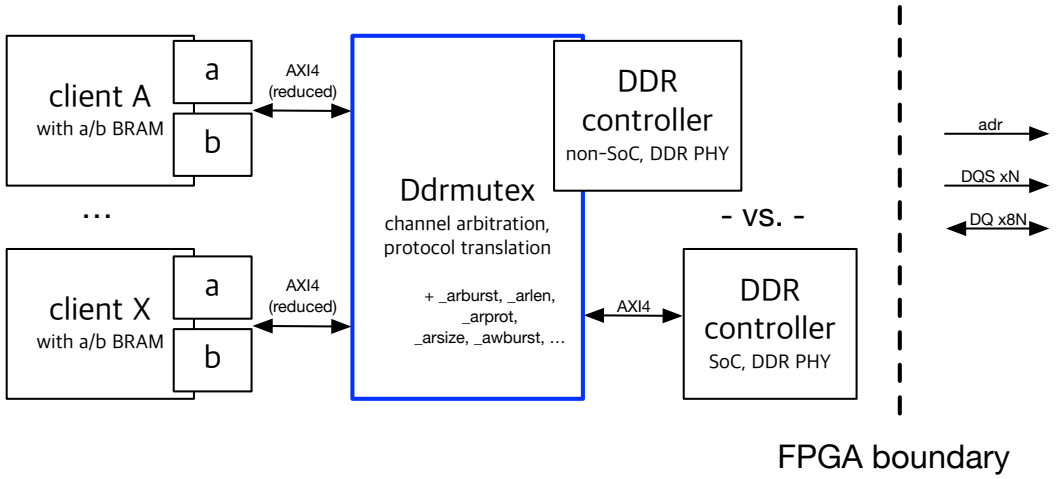
Nice-to-have block structure

Connection to hardware:

- Adressing
- DQS /1, /2, ... (diff.)
- DQ /8, /16, ... (bi-directional)

Connection to RTL:

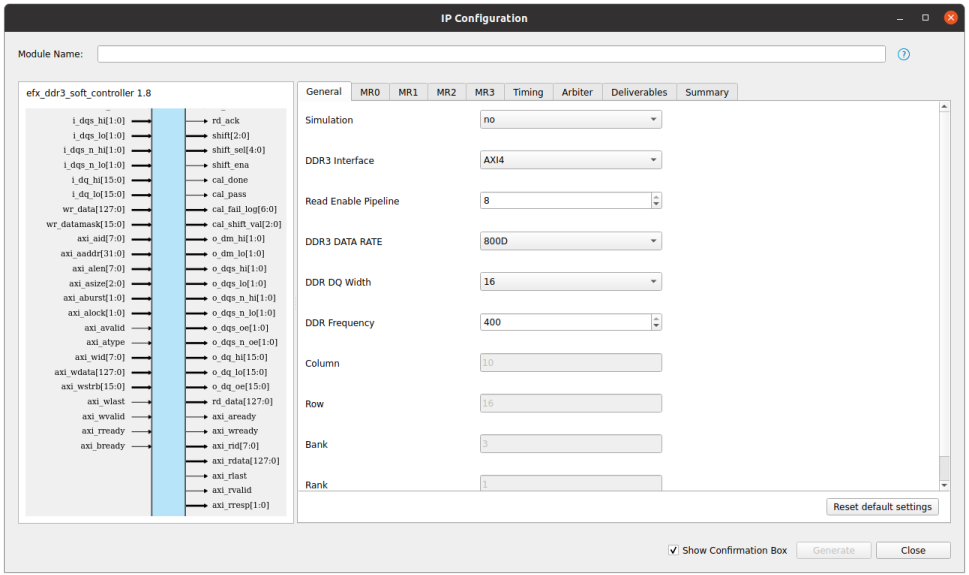
- Set of channels with “reduced“ AXI4 providing start address, configured for burst transfers



DDR Memory

Overview | FPGA integration | Resources

- AMD (non-SoC): AXI4
- Efinix: AXI4

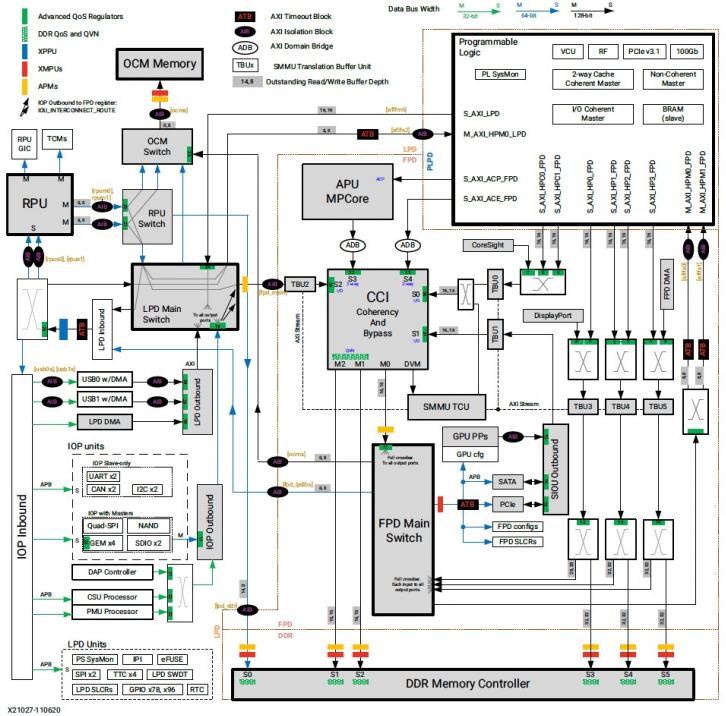


- intelPSG (non-SoC): Avalon

DDR Memory

Overview | FPGA integration | Resources

- The SoC case is special in that the physical memory is shared between CPU complex and FPGA
- Allowable AXI4 burst length may be reduced (e.g. max. 16 beats @128 bit for MPSoC)
- DDR controller configuration typically register mapped in common address space
- Configuration typically through first-stage boot loader
- May need to declare “reserved” section e.g. in Linux device tree



(source: Xilinx UG1085)

DDR Memory

Overview | FPGA integration | Resources

A. Vendor-provided IP documentation, user guides, product guides

1. AMD: Zynq-7000 SoC and 7 Series Devices Memory Interface Solutions (UG586) [IP encrypted, included in free license]
   Zynq-7000 All Programmable SoC TRM (UG585) chapter 10
   Zynq UltraScale+ Device TRM (UG1085) chapter 17
2. Efinix: Titanium DDR DRAM Block (UG-DDR-DRAM-TI)
3. intelPSG: DDR2 SDRAM Controller with UniPHY [IP unencrypted, included in free license]
4. Microchip: PolarFire SoC FPGA DDR Memory Controller (UG0906)

B. Publicly available use cases / examples

C. Relevant work by Open Source community

Remarks

High-speed connectors | Cross-vendor development

Formerly:

- PMOD
- FMC by Xilinx (also adopted by Efinix and Lattice)
- HSMC by Altera

New (low pin count but high-speed):

- SYZYGY, SYZYGY TXR by AMD / OpalKelly (also adopted by Efinix / sort of)
- CRUVI by intelPSG / Trenz

Remarks

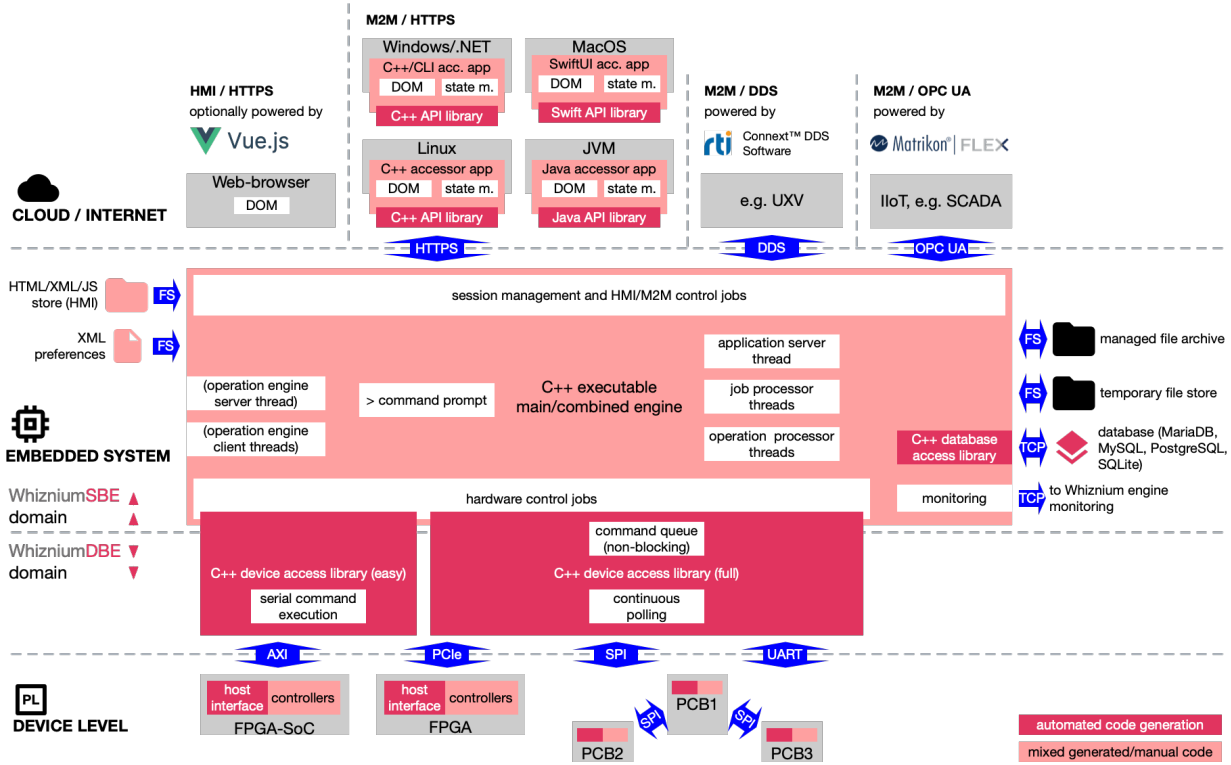
High-speed connectors | Cross-vendor development

- Spend most time outside of vendor IDE’s
- VS Code with standalone GHDL or TerosHDL works really well, beyond syntax checking

- Use vendor IDE’s to extract IP wrappers, work with black boxes as long as possible
- Back to vendor IDE’s for actual synthesis-to-bitstream workflow

Model-based software design with Whiznium

Overview | Concepts | Examples



- Coverage of the “Embedded Full Stack”
- WhizniumDBE (“Device Builder’s Edition”) for FPGA / MCU level and its host access libraries (primary languages: C, VHDL)
- WhizniumSBE (“Service Builder’s Edition”) for Embedded Linux and “outside world” levels (primary languages: C++, HTML)

Model-based software design with Whiznium

Overview | Concepts | Examples

- Successive model composition within an SQL database using import (I) and generation (G) steps
- Output of source code trees only thereafter
- Text-based model files (”diffable”)

WhizniumDBE (Device Builder’s Edition)

- Modular structure (I)
- Command set and buffer transfers (I)
- Data flows and algorithms (I)
- Fine structure (G)
- Custom fine structure (I)
- Finalization (G)

WhizniumSBE (Service Builder’s Edition)

- Deployment information (I)
- Global features (I)
- Database structure (I)
- Basic user interface structure (I)
- Import/export structure (I)
- Operation pack structure (I)
- Custom jobs (I)
- User interface (G)
- Custom user interface features (I)
- Job tree (G)
- Custom job tree features (I)
- Finalization (G)

Model-based software design with Whiznium

- Whiznium is Open Source; the generated code is subject to no license restrictions
- Whiznium generates well-organized, human-readable source code trees which can be synthesized / compiled “out-of-the-box”
- Manual modifications are enabled through the concept of “insertion points”
- Upon source code iteration (e.g. following model extension) manual modifications are carried over to the next version
- Generated code relies on few, well-proven external libraries, all of which are Open Source. Standards are strictly followed
- WhizniumDBE features parametrized “module templates”. Besides corresponding VHDL files, template-specific intervention in the WhizniumDBE master database through C++ code is possible
- WhizniumSBE features parametrized “capability templates”. Also here, template-specific intervention in the WhizniumSBE master database through C++ code is possible

Model-based software design with Whiznium

- For the ability to map and manipulate modular and fine struture of RTL projects, WhizniumDBE is pre-destined for providing the cross-vendor wrappers of the interfaces discussed here
- Many thoroughly tested templates already included: UART/SPI/I^2C, dual-port RAM generator, ...

- Examples for WhizniumDBE + WhizniumSBE interplay
  - Terminal in web-based UI to send commands to / bulk data to/from FPGA
  - Analyzer in web-based UI to track live finite state machine coverage

Model-based software design with Whiznium

Overview | Concepts | Examples

- “Signature“ WhizniumDBE template: host interface, providing RTL and C++ code for host <-> FPGA communication

- From …

fpga	lifcl-40-9bg400c	cleb	Lattice CrossLink-NX Evaluat	true	radiant
	ImelMModule.srefIxVBasetype	hsrefSupRefWdbeMModule	srefTplRefWdbeMModule	sref	Comment
	top		top_v1_0	**top**	
		ImelAMModulePar.x1SrefKKey	Val		
		clkIntNotExt	true		
		extresetNNotP	true		
		fExtclk	50000		
		ImelAMModulePar.end			
		ImelMGeneric.sref	Defval		
		fMclk	50000		
		ImelMGeneric.end			
	ehostif	top	hostif_Easy_v1_0	**hostif**	connection t
		ImelAMModulePar.x1SrefKKey	Val		
		phytype	**uart**		
		fSclk	115200		
		ImelAMModulePar.end			

- vs. -

ImelMUnit.srefIxVB	srefSilRefWdbeMUnit	sref	Title	Easy	srefKToolch
fpga	**xc7z020-1clg400**	**arty**	**Digilent Arty Z7**	**true**	**vivado**
	ImelMModule.srefIxVBasetype	hsrefSupRefWdbeMModule	srefTplRefWdbeMModule	sref	Comment
	top		top_v1_0	**top**	
		ImelAMModulePar.x1SrefKKey	Val		
		clkIntNotExt	false		
		extresetNNotP	true		
		extclkDiffNotSng	false		
		fExtclk	100000		
		ImelAMModulePar.end			
		ImelMGeneric.sref	Defval		
		fMclk	50000		
		ImelMGeneric.end			
	ehostif	top	hostif_Easy_v1_0	**hostif**	
		ImelAMModulePar.x1SrefKKey	Val		
		phytype	**axi**		
		wA	32		
		wD	32		
		ImelAMModulePar.end			

- To …

```
entity Hostif is
  generic (
    fSclk: natural range 100 to 50000000 := 115200;
    fMclk: natural := 50000
  );
  port (
    reset: in std_logic;
    mclk: in std_logic;
    tkclk: in std_logic;
    commok: out std_logic;
    reqReset: out std_logic;

    ...

    reqInvCamacqSetGrrd: out std_logic;
    ackInvCamacqSetGrrd: in std_logic;

    camacqSetGrrdRng: out std_logic_vector(7 downto 0);
    camacqSetGrrdRedNotGray: out std_logic_vector(7 downto 0);

    camacqGetGrrdinfoTixVGrrdbufstate: in std_logic_vector(7 downto 0);
    camacqGetGrrdinfoTkst: in std_logic_vector(31 downto 0);

    ...

    reqPvwabufFromCamacq: out std_logic;
    ackPvwabufFromCamacq: in std_logic;
    dnePvwabufFromCamacq: out std_logic;

    avllenPvwabufFromCamacq: in std_logic_vector(31 downto 0);

    pvwabufFromCamacqAXIS_tready: out std_logic;
    pvwabufFromCamacqAXIS_tvalid: in std_logic;
    pvwabufFromCamacqAXIS_tdata: in std_logic_vector(31 downto 0);
    pvwabufFromCamacqAXIS_tlast: in std_logic;

    ...

    rxd: in std_logic;
    txd: out std_logic
  );
end Hostif;
```

command camacq.setGrrd

preview buffer transfer

UART

- vs. -

```
entity Hostif is
  generic (
    fMclk: natural := 50000
  );
  port (
    reset: in std_logic;
    mclk: in std_logic;
    tkclk: in std_logic;
    commok: out std_logic;
    reqReset: out std_logic;

    ...

    reqInvCamacqSetGrrd: out std_logic;
    ackInvCamacqSetGrrd: in std_logic;

    camacqSetGrrdRng: out std_logic_vector(7 downto 0);
    camacqSetGrrdRedNotGray: out std_logic_vector(7 downto 0);

    camacqGetGrrdinfoTixVGrrdbufstate: in std_logic_vector(7 downto 0);
    camacqGetGrrdinfoTkst: in std_logic_vector(31 downto 0);

    ...

    reqPvwabufFromCamacq: out std_logic;
    ackPvwabufFromCamacq: in std_logic;
    dnePvwabufFromCamacq: out std_logic;

    avllenPvwabufFromCamacq: in std_logic_vector(31 downto 0);

    pvwabufFromCamacqAXIS_tready: out std_logic;
    pvwabufFromCamacqAXIS_tvalid: in std_logic;
    pvwabufFromCamacqAXIS_tdata: in std_logic_vector(31 downto 0);
    pvwabufFromCamacqAXIS_tlast: in std_logic;

    ...

    AXIL_araddr: in std_logic_vector(31 downto 0);
    AXIL_arprot: in std_logic_vector(2 downto 0);
    AXIL_arready: out std_logic;
    AXIL_arvalid: in std_logic;
    AXIL_rdata: out std_logic_vector(31 downto 0);
    AXIL_rready: in std_logic;
    AXIL_rresp: out std_logic_vector(1 downto 0);
    AXIL_rvalid: out std_logic;
    AXIL_awaddr: in std_logic_vector(31 downto 0);
    AXIL_awprot: in std_logic_vector(2 downto 0);
    AXIL_awready: out std_logic;
    AXIL_awvalid: in std_logic;
    AXIL_wdata: in std_logic_vector(31 downto 0);
    AXIL_wready: out std_logic;
    AXIL_wstrb: in std_logic_vector(3 downto 0);
    AXIL_wvalid: in std_logic;
    AXIL_bready: in std_logic;
    AXIL_bresp: out std_logic_vector(1 downto 0);
    AXIL_bvalid: out std_logic
  );
end Hostif;
```

AXI lite /32

Conclusion

- Success in working with the new low-cost FPGA interfaces still is hard earned
  - Non-standardized hardware / connectors are one obstacle
  - Complexity / non-open IP of vendors are another obstacle

- Unifying concepts such as used in Linux kernel + device tree are missing in the FPGA world

- Model-based source code generation can help
  - WhizniumDBE comes with tested cross-vendor modules and will also include interface wrappers
  - WhizniumDBE maintains a coarse-to-fine project model in a database and is user-extensible (by means of C++ code, e.g. for frequently used IP)

Thank You!

Questions?

Also, feel free to connect.

- https://www.linkedin.com/in/wirthmua
- https://github.com/mpsitech

Alexander Wirthmüller
Founder & Director

Phone: +49 (89) 4524 3826
Mobile: +49 (175) 918 5480
E-Mail: aw@mpsitech.com

MPSI Technologies GmbH
Agnes-Pockels-Bogen 1
80992 Munich
Germany
www.mpsitech.com