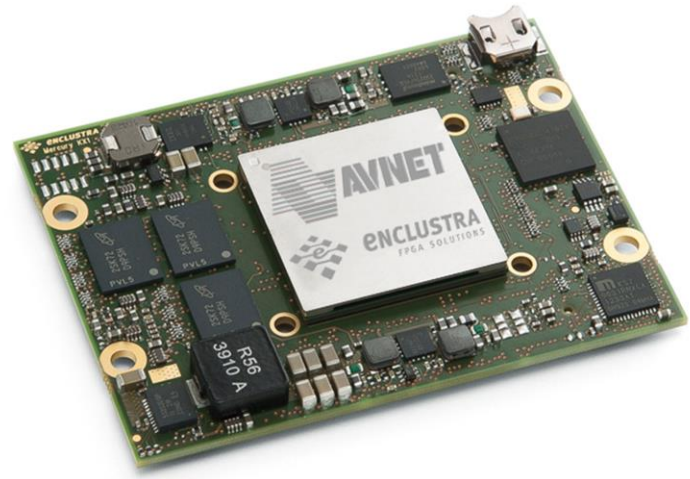
ENCLUSTRA FPGA SOLUTIONS

CPU Offloading using FPGA Fabric

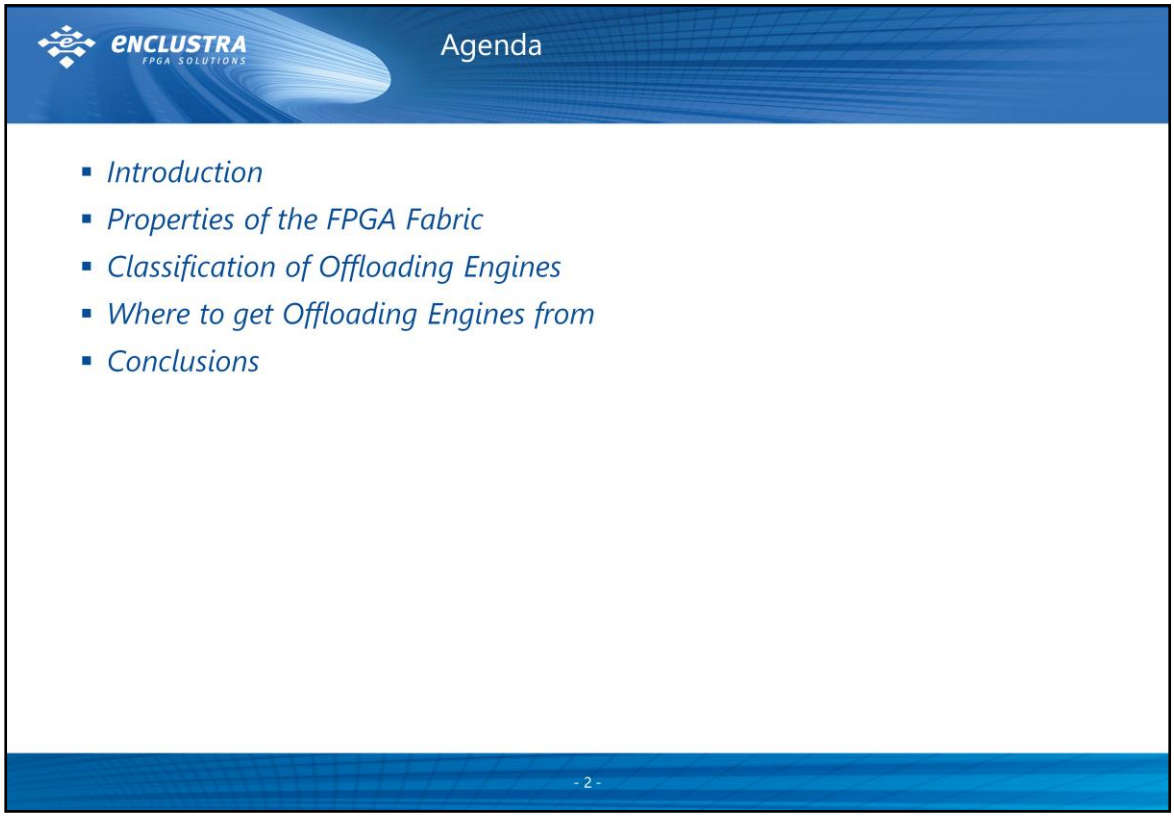


Avnet Silica & Enclustra Seminar

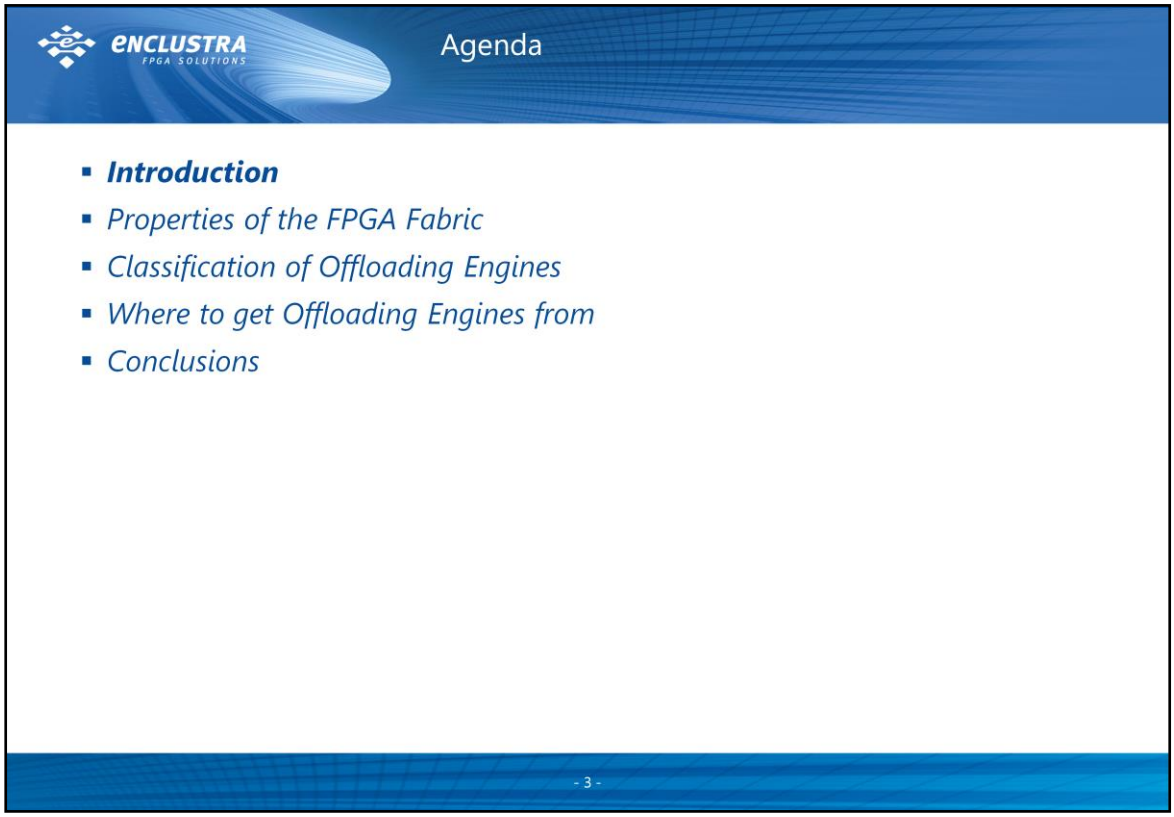
"Getting started with Xilinx Zynq SoC"

Fribourg, April 26, 2017

Oliver Bründler
Project Leader
Enclustra GmbH

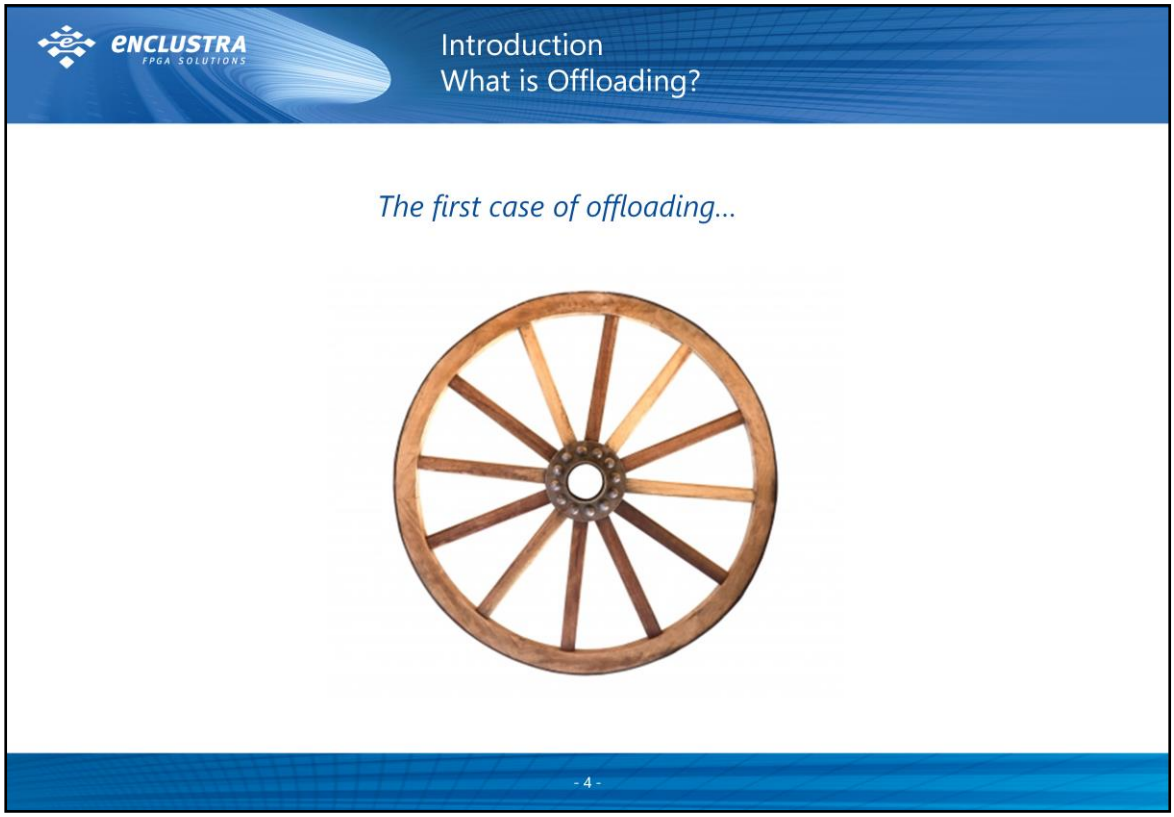


Enclustra GmbH
Räffelstrasse 28 – CH-8045 Zürich – Switzerland
Phone +41 43 343 39 43 – www.enclustra.com



Introduction

The next few slides give a short introduction of what CPU offloading is and how it can help improving system performance.



What is Offloading?

Offloading means taking load from one instance and moving it to another. There are several reasons for doing so. The most common examples are:

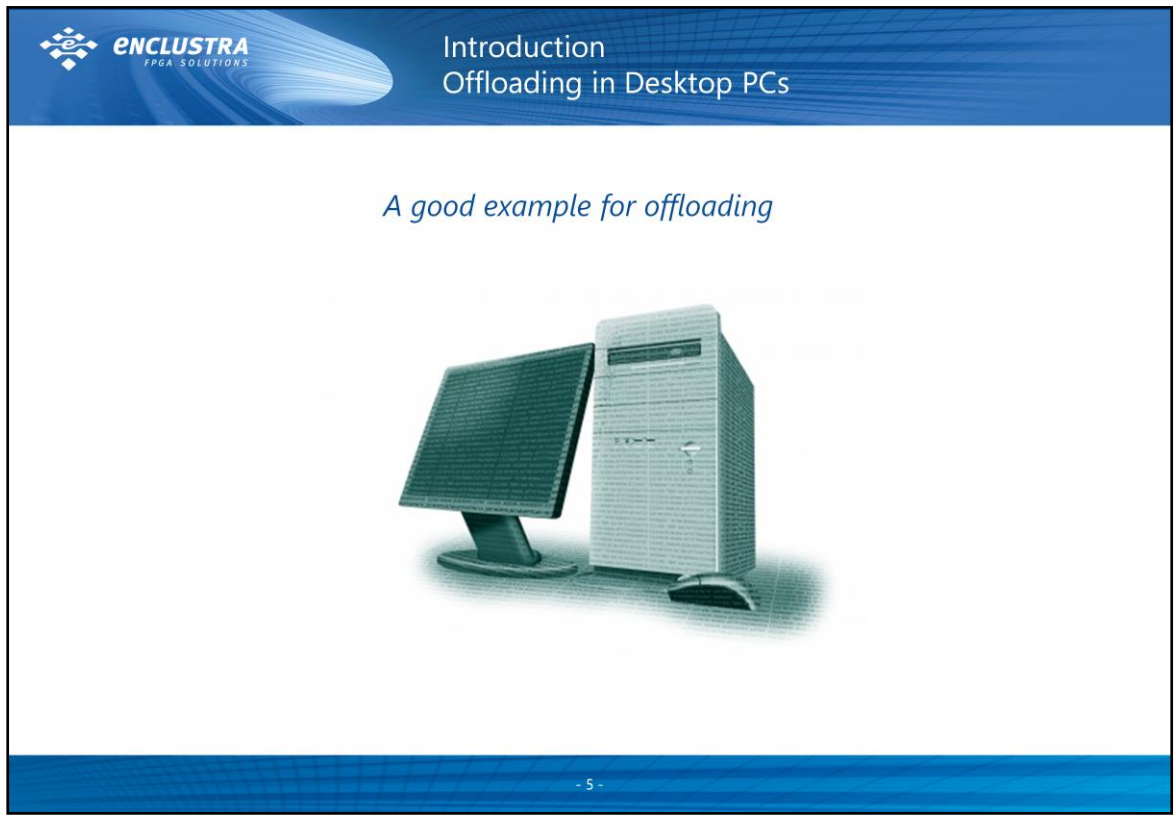
- One instance can do the same work with less effort than the other
- One instance can do more work of a certain type faster than the other
- One instance must be made available for other tasks

As an example we take the potentially first occurrence of offloading in mankind's history: the invention of the wheel.

Using a wheel to transport heavy load is by far more energy efficient than carrying the load by hand. It also frees a very important and flexible resource (the human) for more important tasks (such as finding the direction to the destination) by using a less flexible resource (the wheel). Thanks to its specialization, the wheel is able to carry things that are way too heavy to be carried by a man.

This example nicely shows what offloading is and – historically proven – that it makes sense.

Enclustra GmbH
Räffelstrasse 28 – CH-8045 Zürich – Switzerland
Phone +41 43 343 39 43 – www.enclustra.com



Offloading in Desktop PCs

Offloading in electronics and computer design is nothing new as well. It was already present in the Commodore C64 where graphics and audio chips strongly offloaded the CPU.

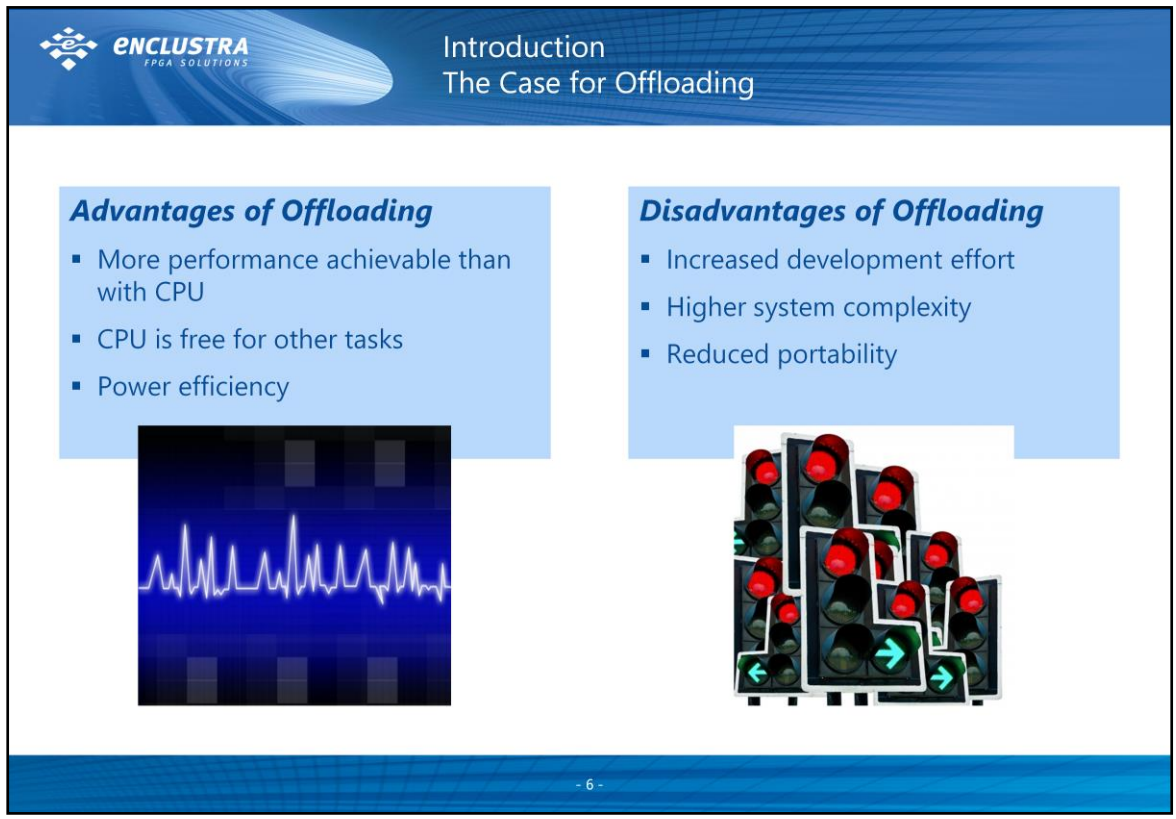
A few examples of CPU offloading that happens in a standard desktop PC are:

- Checksum calculations done by network cards
- DMA engines for copying data
- Graphics offloading by GPUs
- Sound cards
- Error correction done directly in hard drive
- Speed control of the DVD drive
- ...

In fact all the offloading engines running in a typical desktop PC outperform the processing power of the CPU by factors. So why are we so focused on the CPU if it comes to buying a powerful PC? The answer is quite simple: The CPU is the most flexible part of a PC. All those offloading engines are relatively dumb and targeted to just doing one thing while the CPU is flexible enough to run a vast variety of algorithms. The CPU is the part of the PC that is easily programmable and therefore the main reason for the success of the PC architecture as well. Most of the offloading engines are not accessible and can therefore not be used to run custom software.

On the other hand the offloading engines can execute their relatively simple tasks way faster than the CPU with less power consumption and keep the CPU free for executing our programs.

Enclustra GmbH
Räffelstrasse 28 – CH-8045 Zürich – Switzerland
Phone +41 43 343 39 43 – www.enclustra.com



Advantages and Disadvantages of Offloading

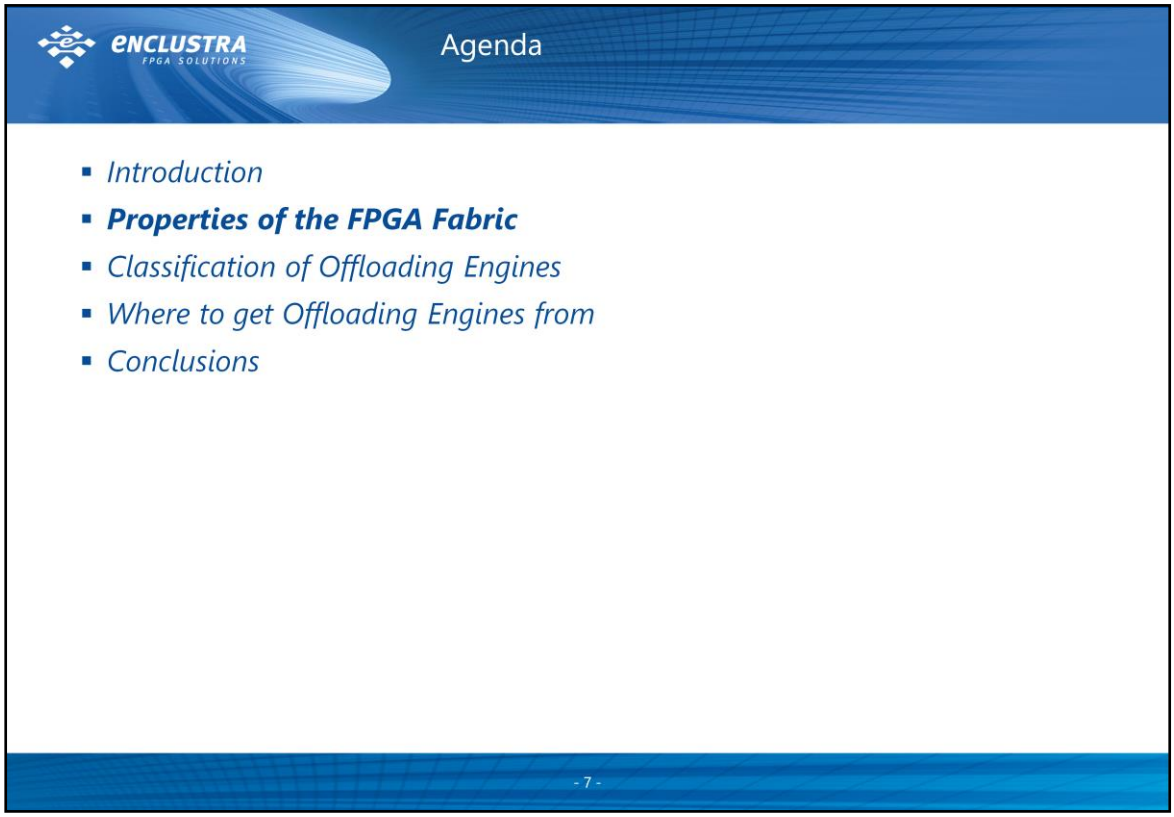
While offloading is a powerful concept, it is certainly not the ideal solution to all problems. Offloading is generally an optimization technique and - exactly as with all optimization techniques - it should only be applied if it fits the problem. For example does it not make sense to do any offloading for the status change of a traffic light.

- The CPU load in a traffic light is rather uncritical
- There is no benefit for the end application
- The engineering effort would increase
- Cost would increase since a CPU with any kind of offloading engine is more expensive than same CPU alone
- System complexity increases

On the other hand, there are applications where offloading can boost the performance of a system. A good example for this category are signal processing systems.

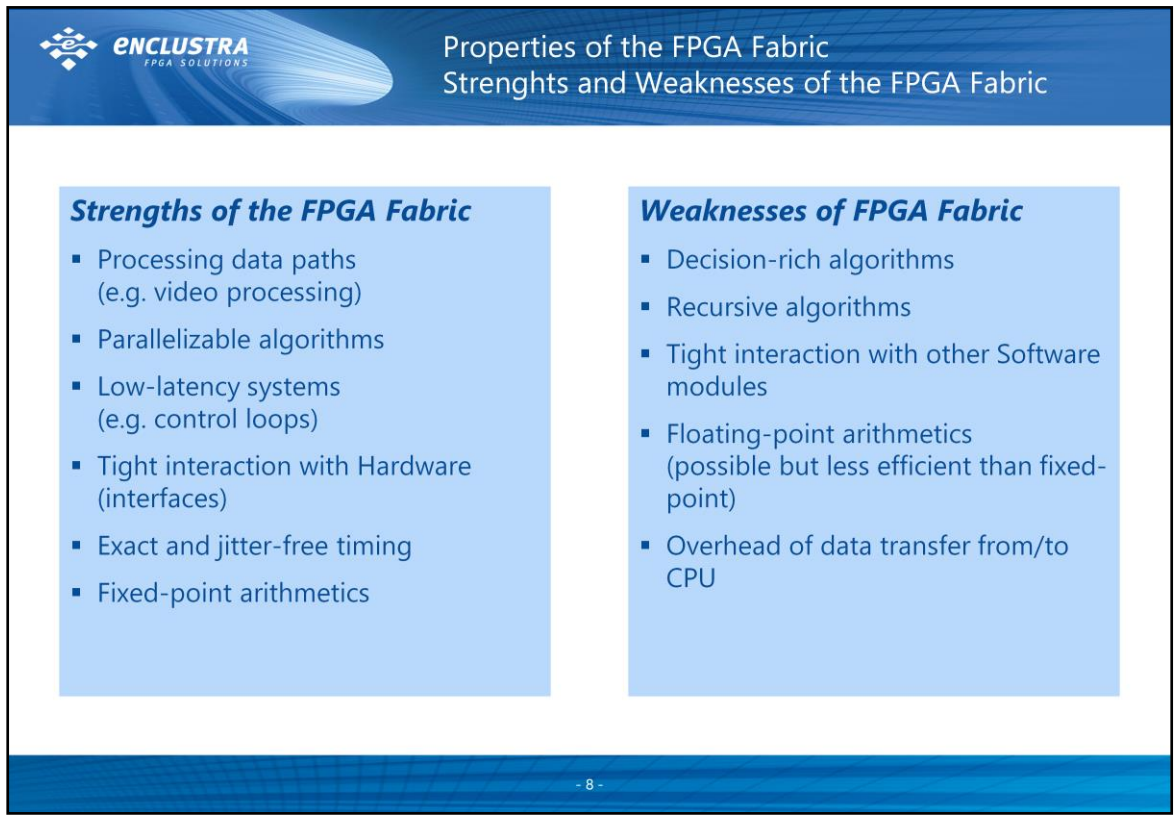
- Offloading engines can reduce the CPU load for operations such as filtering close to zero
- The performance can be increased which is beneficial in many applications (better filters can be implemented)
- Power is reduced since an offloading engine is more efficient than the CPU for a specific task
- The cost compared to a (multi-)CPU setup with the required performance is often reduced

The example illustrates that offloading is not the ideal solution to all problems. So in general it is recommendable to find out whether CPU offloading is of any use for your application, decide which tasks benefit the most from offloading and implement offloading only for these parts of your application.



Properties of the FPGA Fabric

To find out whether offloading using the FPGA fabric fits a given application, a minimal knoweldge about the properties of the FPGA fabric is required. This section gives a very brief overview without going into too much details. If you are interested in details of designing circuitry using the FPGA fabric, an FPGA development seminar is the event of your choice.



Strengths and Weaknesses of the FPGA Fabric

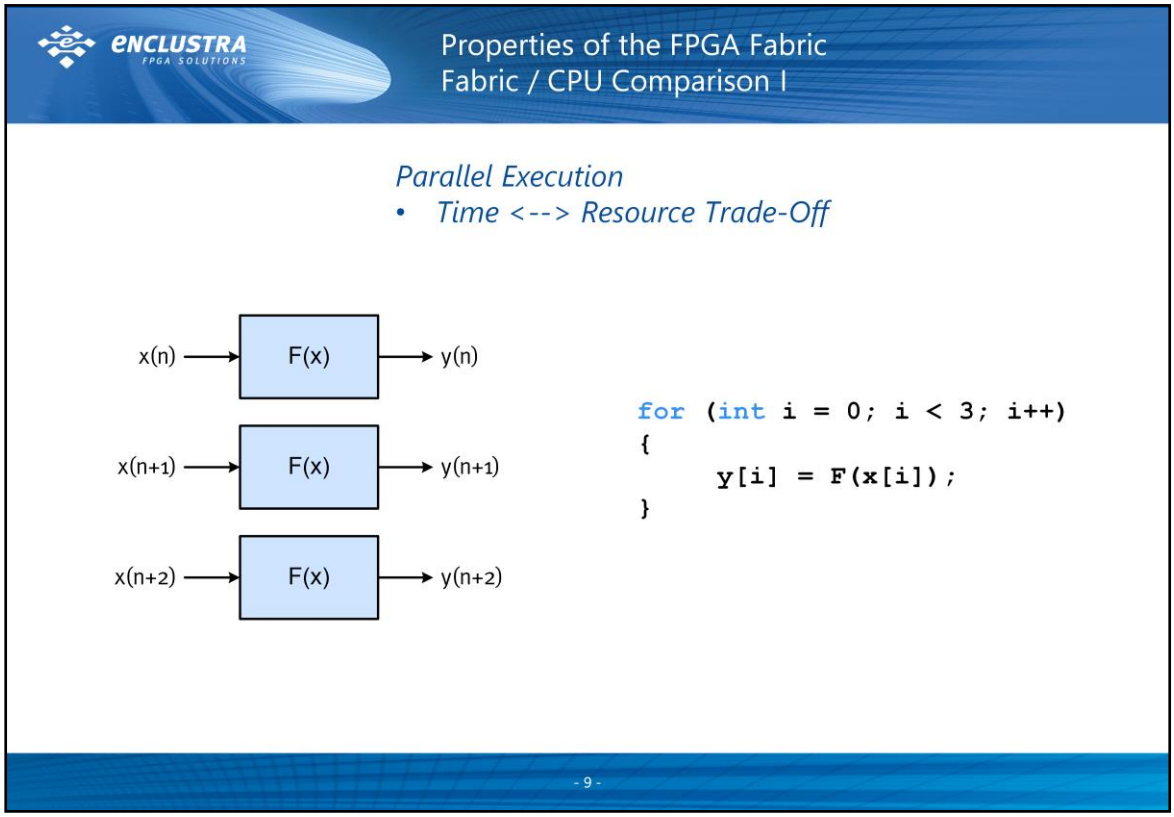
From the last slide we now know the general advantages and disadvantages of offloading. To decide whether an SoC[1] really fits your application, it is also important to know the specific strengths and weaknesses of offloading using the FPGA fabric.

Please note that the items mentioned in the slide above are meant more in a relative than in an absolute way. It is for example certainly possible to implement a reasonable offloading engine for a decision-rich algorithm but it will require more effort and will be less beneficial than accelerating a signal processing data path.

Another example for the relativeness of the items mentioned is the exact and jitter-free timing of FPGA fabric. This is also achievable with software with additional effort but out-of-the-box software is affected by jitter of the scheduler, interrupts and different latencies of different processing paths.

Some background for the items listed in the slide above is provided in the next slides.

[1] System on chip, combination of CPU and FPGA fabric on the same die



Parallel Execution

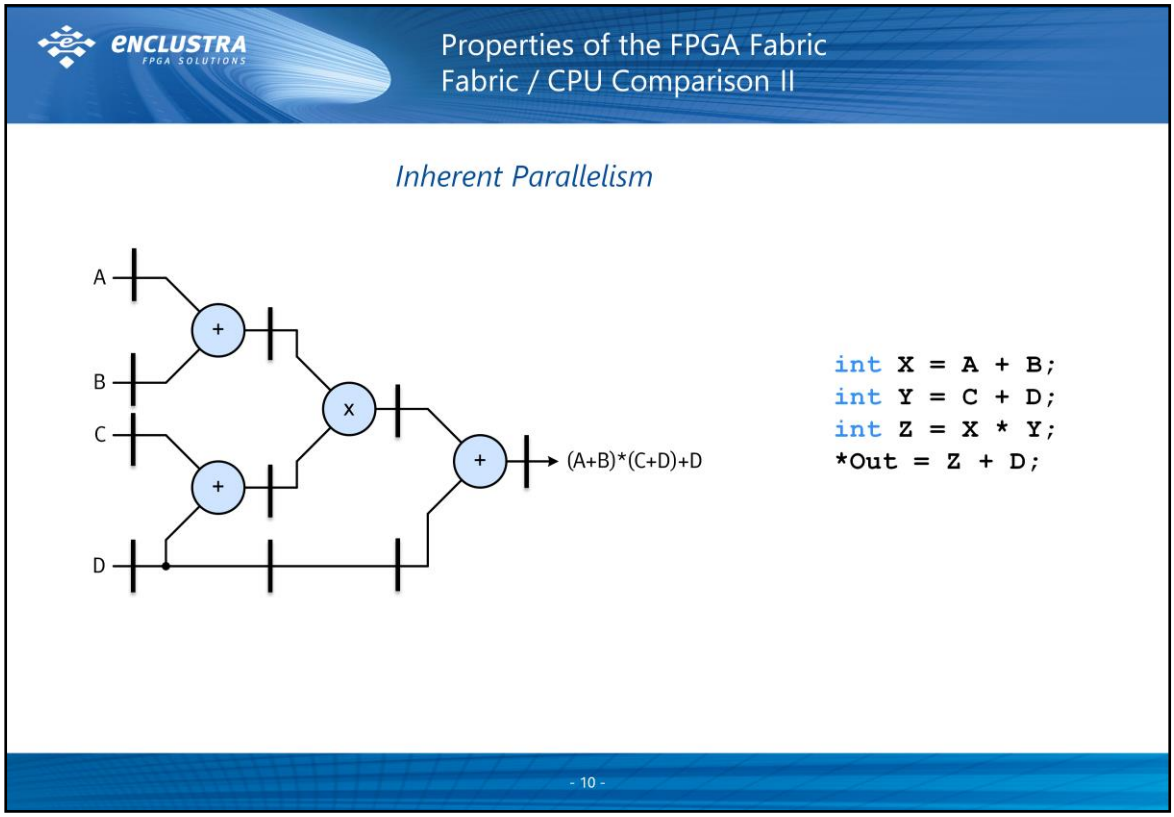
Algorithms that execute the same computations on different data sets individually are perfectly suited for CPU offloading by the FPGA fabric. While a CPU needs to execute one computation after the other, it is possible to do multiple computations in parallel in the FPGA fabric.

A good example for this is brightening an image, which means adding a value to every pixel of the image. While a CPU has to execute the addition for every pixel one after the other, an offloading engine implemented in the FPGA fabric can execute the addition for multiple pixels in parallel.

Of course this approach has its limitations as well. Let's assume the image from the example above has 1M pixels. In that case it would not be reasonable to do all additions in parallel. On one hand because of the huge amount of resources this would require, on the other hand because 99% of the theoretical performance would be wasted by the bottleneck of getting the image into and out of the offloading engine. A more reasonable approach would be to only partially parallelize the calculations (e.g. calculate 16 pixels in parallel).

In general, the number of parallel computations can be used to trade-off performance versus resource usage.

Enclustra GmbH
Räffelstrasse 28 – CH-8045 Zürich – Switzerland
Phone +41 43 343 39 43 – www.enclustra.com



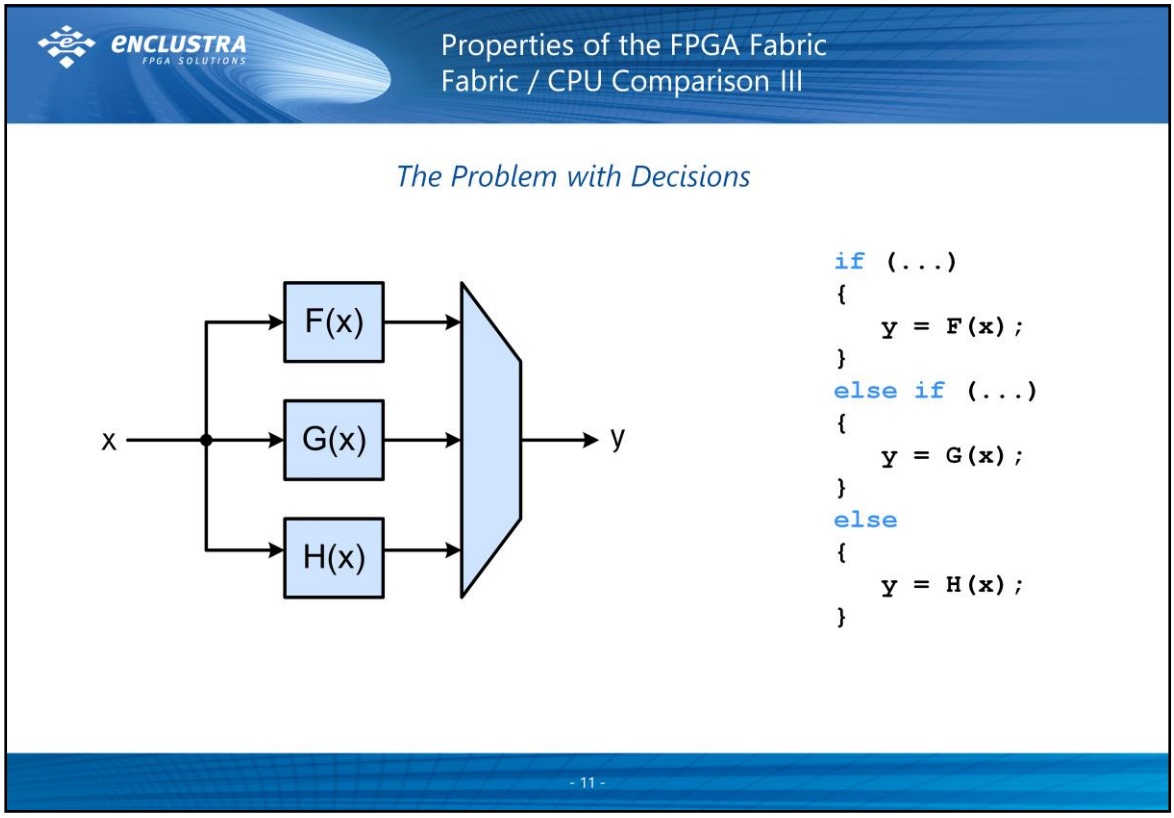
Inherent Parallelism

Parallelization is not only applicable to complete processing steps but also to different operations within one processing step. Since the FPGA fabric inherently executes every operation in parallel, complex calculations can be executed every clock cycle while a CPU requires at least one clock cycle for every operation.

Of course the FPGA fabric is not infinitely fast and operations also take some time, even if they can happen in parallel. If we assume every operation of the example shown to require one clock cycle, we have to add registers at the input and output of every operation to make sure the operation is completed before the data is stored in a register. This increases the latency to 4 clock cycles, so the advantage over the CPU is lost, right?

No! while the latency increases to 4 clock cycles, it is still possible to apply new data every clock cycle and receive a new result every clock cycle. This nicely shows the difference between latency (4 clock cycles) and insertion delay (1 clock cycle).

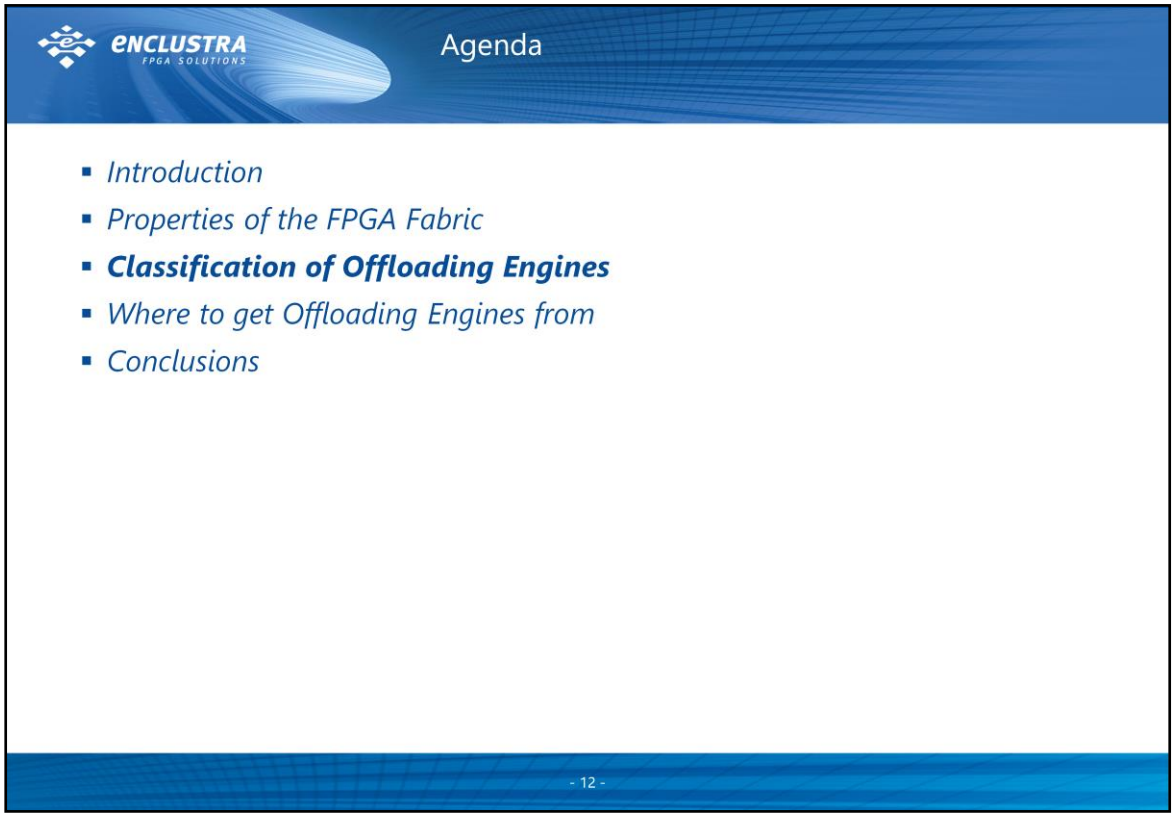
For CPUs the latency and the insertion delay are always the same because they can only start a new calculation if the previous one has been completed. Therefore the differentiation between latency and insertion delay may seem a bit confusing to software developers at the beginning.



The Problem with Decisions

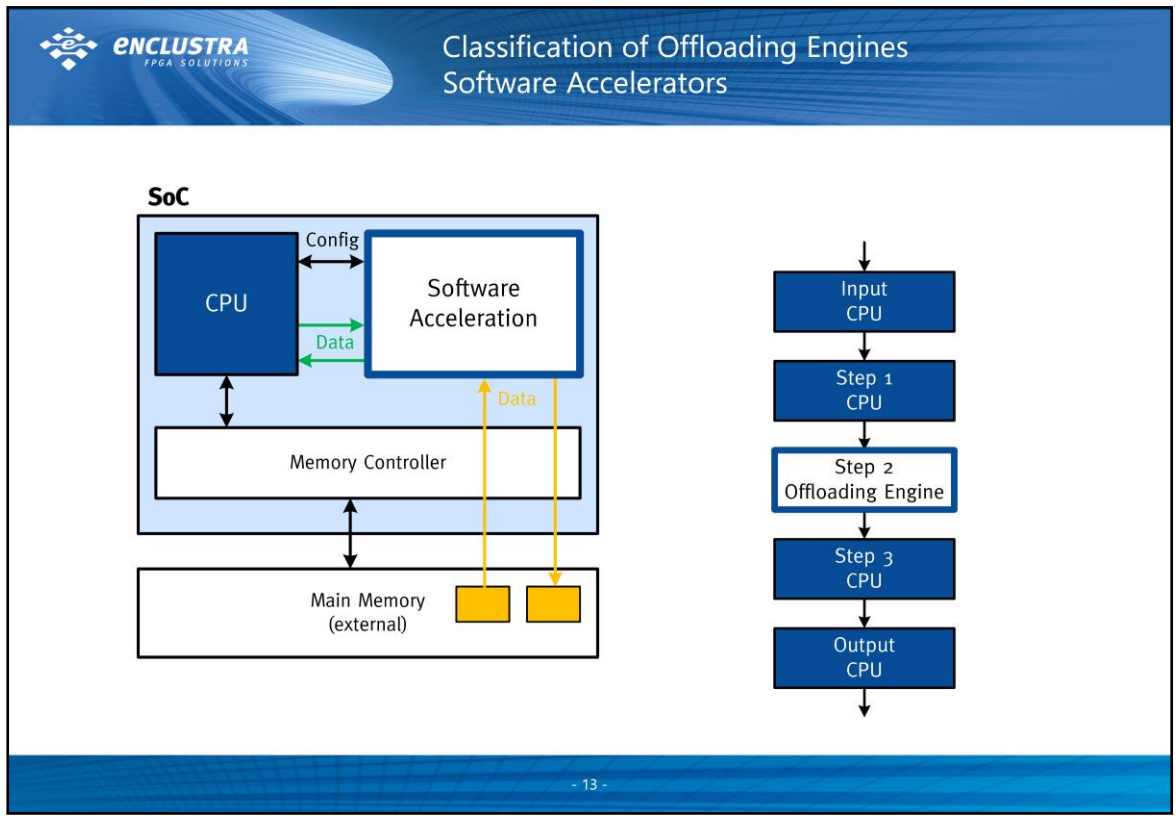
It was already mentioned that decision-rich algorithms are not a perfect fit for acceleration using the FPGA fabric. The reason for this is that hardware resources are required for every possible branch. In software, CPU time is only spent on the branch that is actually executed. As a result, decision-rich algorithms tend to be very resource intensive to accelerate.

Nevertheless, such algorithms can be accelerated and experienced FPGA/SoC engineers are often able to reduce the impact of this fact by clever design and resource sharing between different calculation branches.



Classification of Offloading Engines

The next few slides will give an overview over different classes of offloading engines along with some design considerations and examples for each class. In reality many offloading engines do not fit exactly into one class but combine two or more of them, therefore some classification examples for real-world applications are given at the end of this section. Note that there are no widely used naming conventions, so the naming in literature may differ from the naming in this presentation.



Description

We use the name «Software Accelerator» for any offloading engine whose only purpose is to speed up the execution of a software and/or reduce the CPU load for doing so. Therefore input to and output from a software accelerator come from and go to software (i.e. the CPU).

Besides the data input/output there is also a configuration interface present allowing the software to configure the offloading engine to execute exactly the operation required. This is the case for all types of offloading engines and not mentioned sepearately in each of the following slides.

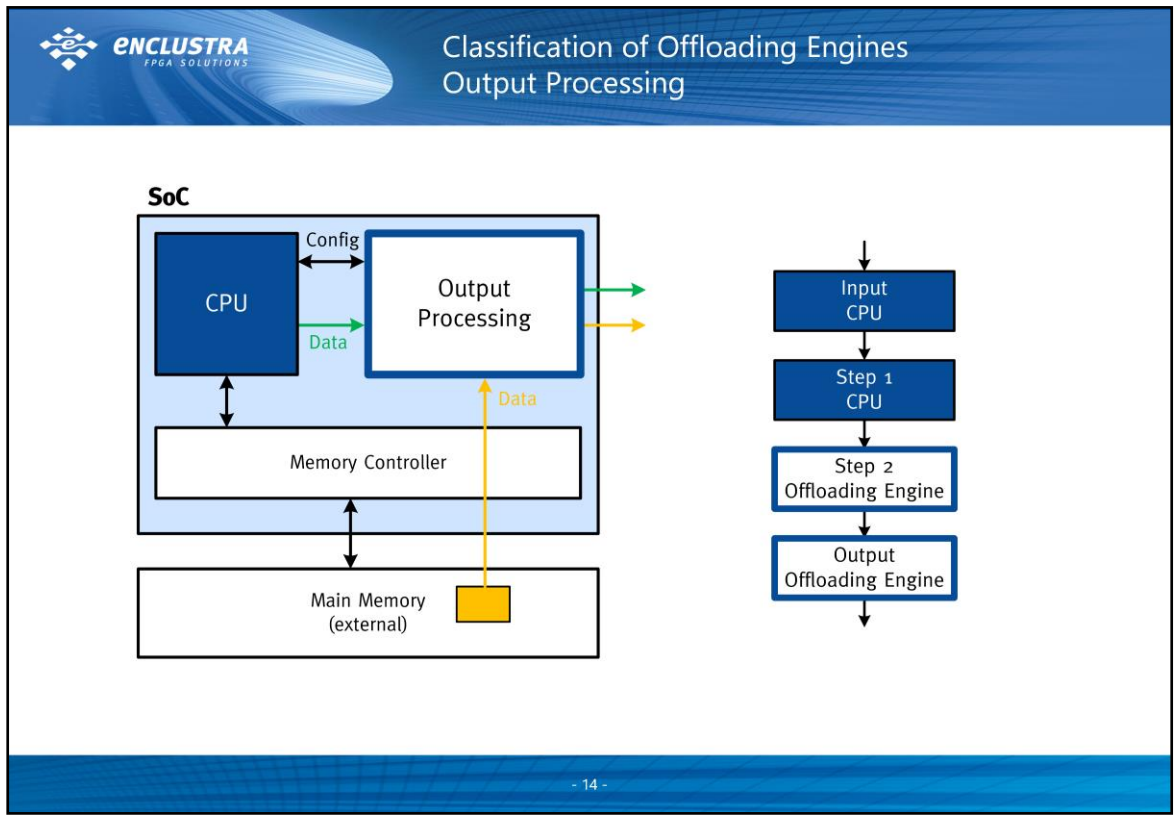
DMA engines are a special case of software accelerators: They accelerate a piece of Software that copies data without performing any operations on them.

Design Considerations

- If the offloading engine operates on large amounts of data, it is recommended to read/write the data directly from/to main memory. Otherwise significant CPU power would be wasted for transferring data between CPU and offloading engine.
- Input and output can be asymmetric. If the offloading engine strongly reduces the amount of data, it is reasonable to read data directly from the main memory but let the CPU read out the result directly. The opposite is also possible.
- To maximize performance, the system architecture should allow using the CPU for other tasks while the offloading engine is running.

Examples

- FFT calculation (memory-in, memory-out)
- Center of gravity calculation over an image (memory-in, CPU-out)
- Equation solving algorithm (CPU-in, CPU-out)



Description

We use the name «Output Processing» for any offloading engine that is playing out data generated by the CPU to some other device. In many cases the offloading engine not only plays out data but also executes some processing on it.

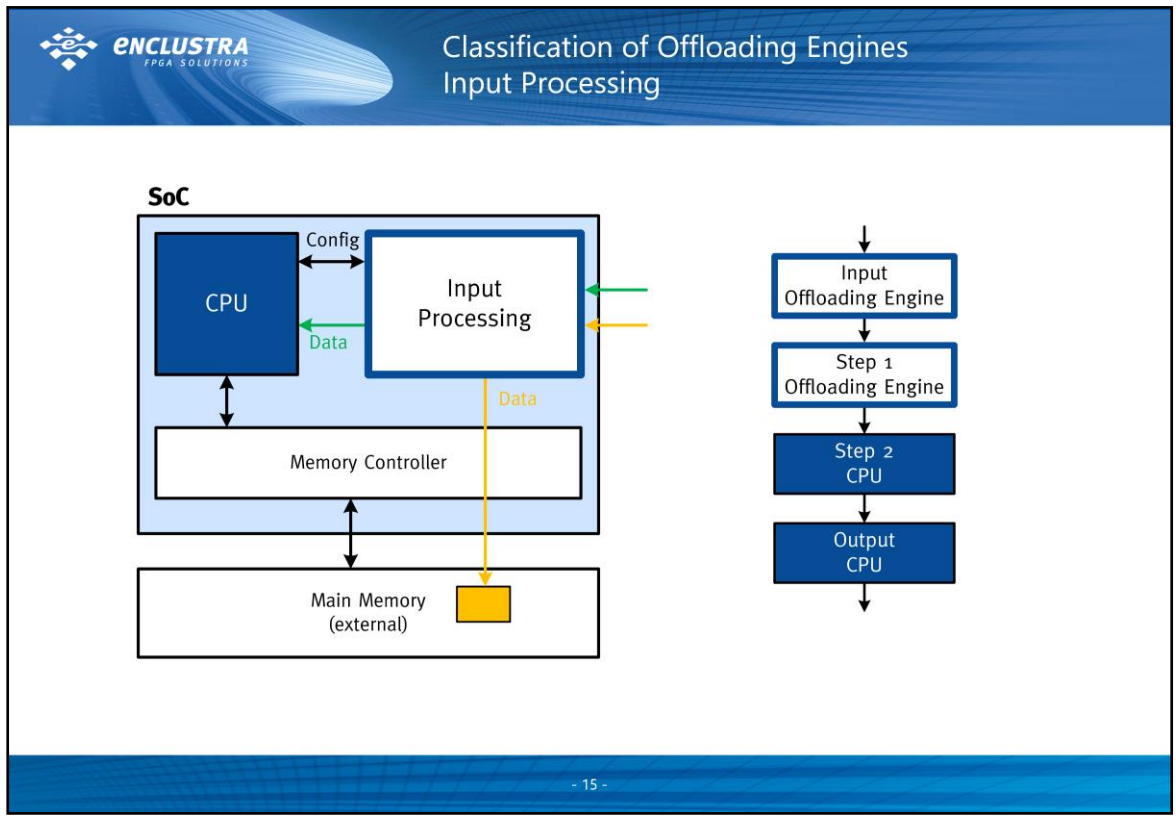
One requirement for many output processing engines is to ensure a constant output data rate while the CPU generates the data offline at a completely different and often inconstant rate.

Design Considerations

- If the input bandwidth is considerable, it is recommended to read the data directly from the main memory to reduce the CPU power required for transferring data to the offloading engine.
- While the offloading engine can generate exact timing, the software is often affected by jitter. It is therefore most often required to implement a buffer within the offloading engine to compensate for the jitter of the software.
- Underrun conditions can occur if the CPU is not able to provide enough data in time and should therefore be detected and handled.
- For applications that expand/interpolate data, it is preferable to do the interpolation in the offloading engine to reduce the amount of data to be transferred from the CPU to the offloading engine.
- The latency is often not critical for output processing engines.

Examples

- Audio output at an exact rate (memory-in, hardware-out)
- RF packet transmission including protocol handling and modulation (CPU-in, hardware-out)



Description

The name «Input Processing» is used for the inverse of the output processing. It is used for an offloading engine that captures input data and often also does some pre-processing on it before the data is passed to the CPU.

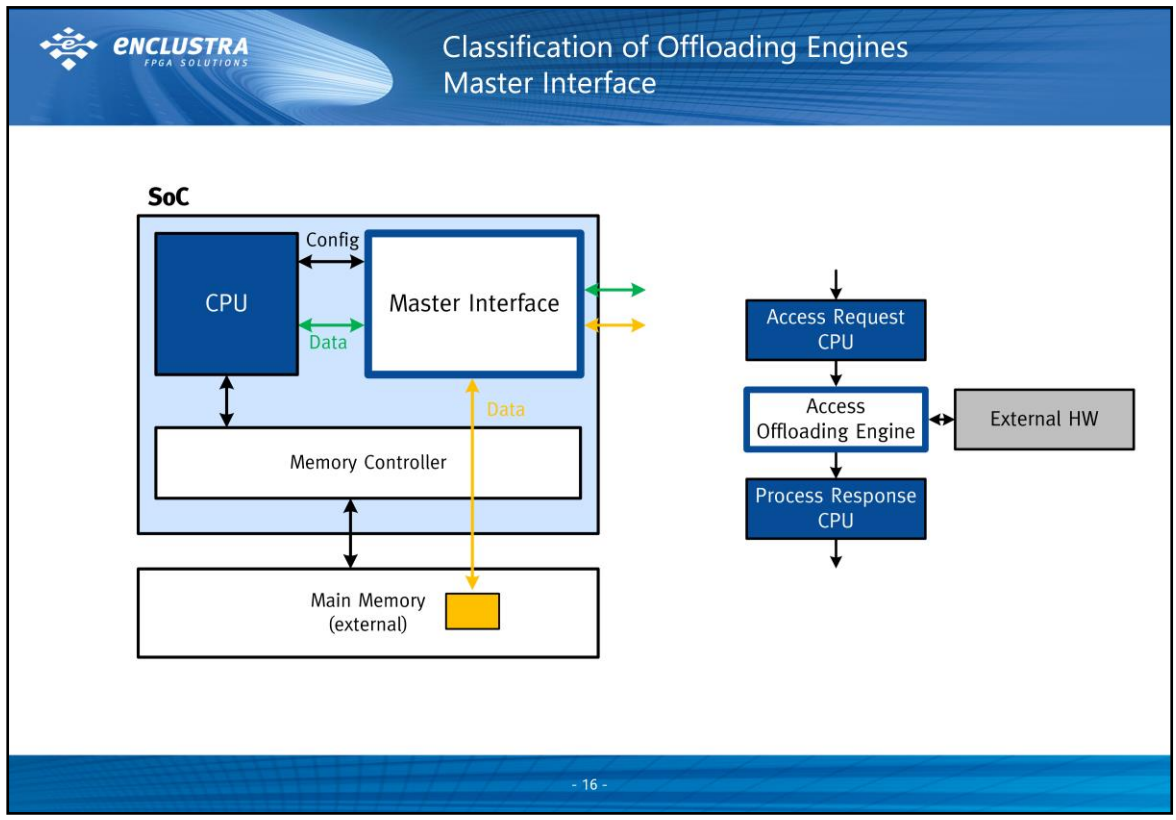
Exactly as output processing engines, input processing engines are often used to build a bridge between fixed sample rates of hardware components and block-processing done by the CPU.

Design Considerations

- If the captured input data cannot be compressed, it is recommended to write the data directly into the main memory to reduce CPU load.
- While the input is usually arriving at a constant data rate, the software is affected by jitter. It is therefore very often required to implement a buffer within the offloading engine to compensate for the jitter of the software.
- Overrun conditions can occur if the CPU is not able to process enough data in time and should therefore be detected and handled.
- Wherever it is possible to compress (decimate) the data captured, this should be done within the offloading engine.
- The latency is not critical for input processing engines in many cases.

Examples

- Video capture and filtering (hardware-in, memory-out)
- Sigma-Delta-ADC down-conversion (hardware-in, CPU-out)



Description

«Master Interfaces» are mentioned as offloading engines here, even though they are often called peripherals in everday-work. However, they prevent the CPU from having to do bit-banging and as result offload the CPU, they can be considered as offloading engines.

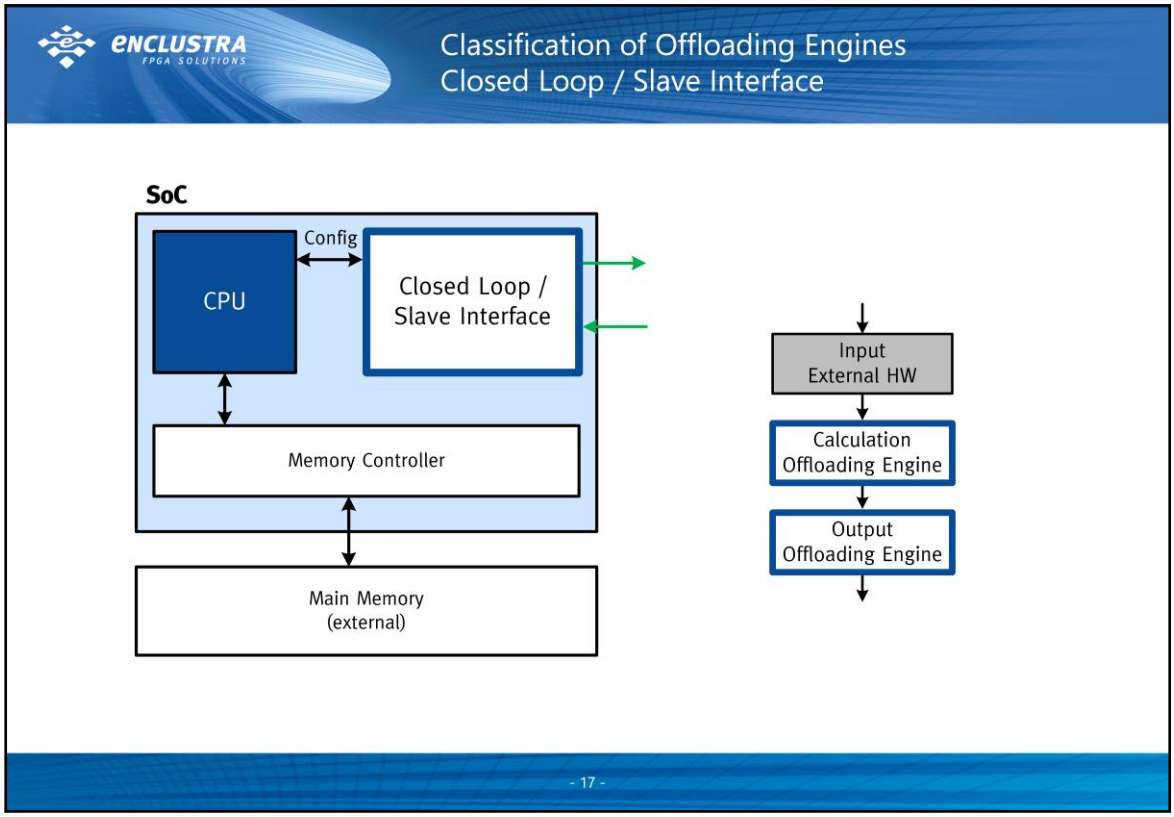
This class of offloading engines is explicitly mentioned in the presentation since the ability to implement even the most exotic interfaces to any external components is an important feature of SoCs.

Design Considerations

- The latency is often critical for master interfaces, since the CPU may wait for completion of an access to an external component until it can continue its operation.
- In many cases the IRQ load of the CPU can be reduced by adding more intelligence to the interfaces.

Examples

- DDR memory controller for special requirements
- SDIO controller



Description

«Closed Loop» and «Slave Interface» offloading engines are similar, since actions of both of them are triggered by an external input. Without using an offloading engine, an IRQ would be required and the CPU would have to react on the input. Compared to this, an offloading engine can react faster and jitter free.

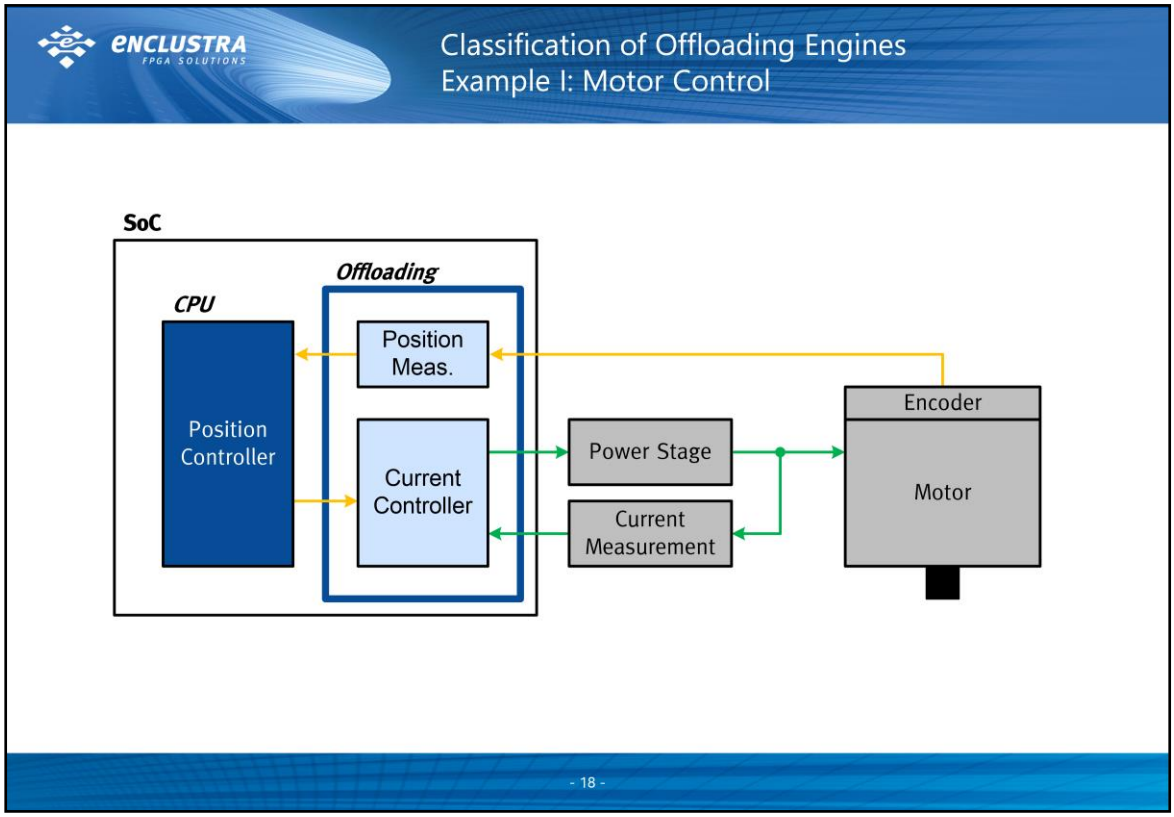
Especially the closed loop case is important since FPGAs are often used in control applications.

Design Considerations

- In closed loop systems, the latency is usually critical. This is true especially for control loops since additional latency degrades stability of the loop.
- Known and fixed latency is also often essential because jitter also reduces stability of control loops.
- For closed loop offloading engines, the CPU interaction should be kept minimal when they are running. It is usually reasonable to only notify the CPU if any unexpected conditions occur.

Examples

- Motor control loop (mostly current control)
- Slave interface connected to an external processor
- Ethernet packet switch/filter



Example 1: Motor Control

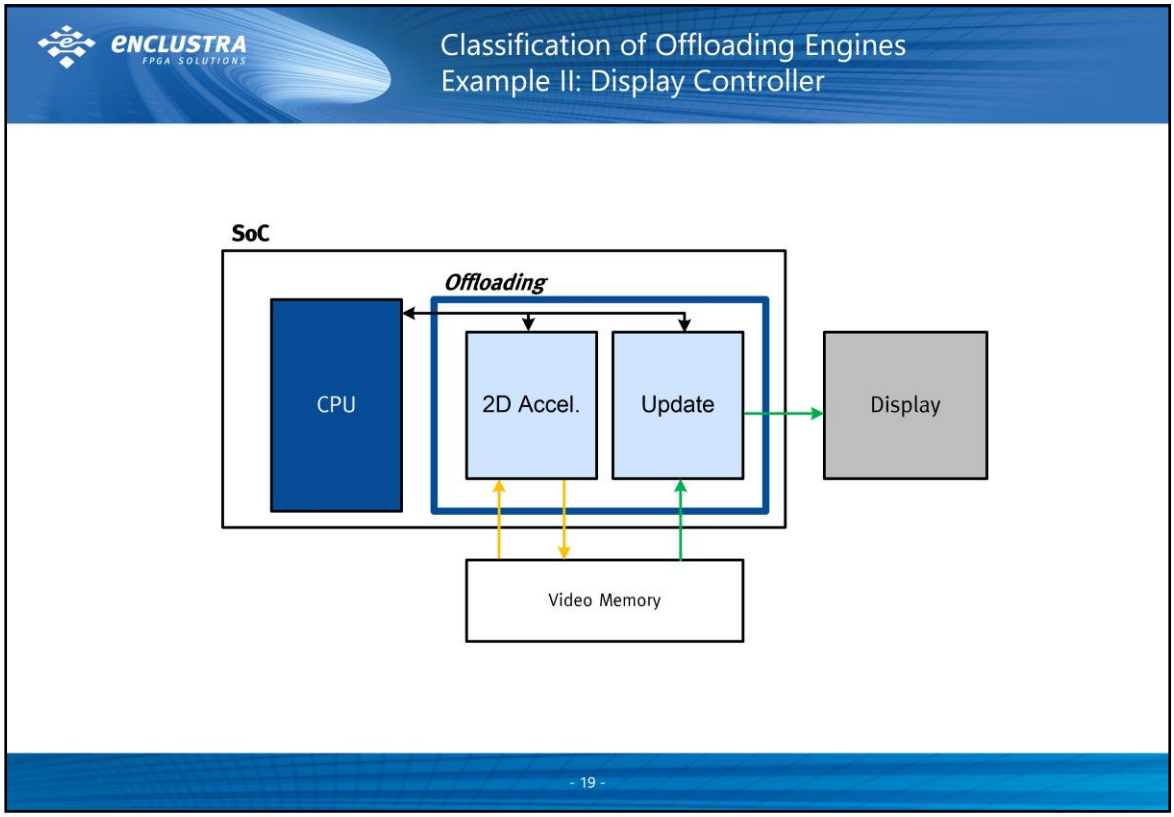
Usually offloading engines do not just fit into one of the classes presented but they combine two ore more of them.

A good example for this is a motor control system with a fast but relatively simple current controller (PID, ~200 kHz) realized directly in the FPGA fabric and a relatively slow but complex position controller (state-based, ~5 kHz) realized in the CPU software.

In this system the current control loop clearly fits into the category «Closed Loop». In the position control loop, the position measurement can be seen as «Input Processing» and the current control loop as a whole can be regarded as «Output Processing» since it just applies a given current value to the motor.

In this case the CPU accesses the offloading engine directly since only single values must be read/written.

It is obvious that the offloading engine belongs to three categories. This example also nicely shows that system architecture overrules general assumptions: While latency usually is not critical to input and output processing engines, in this case it is because the CPU closes a control loop containing the position measurement and the current controller.

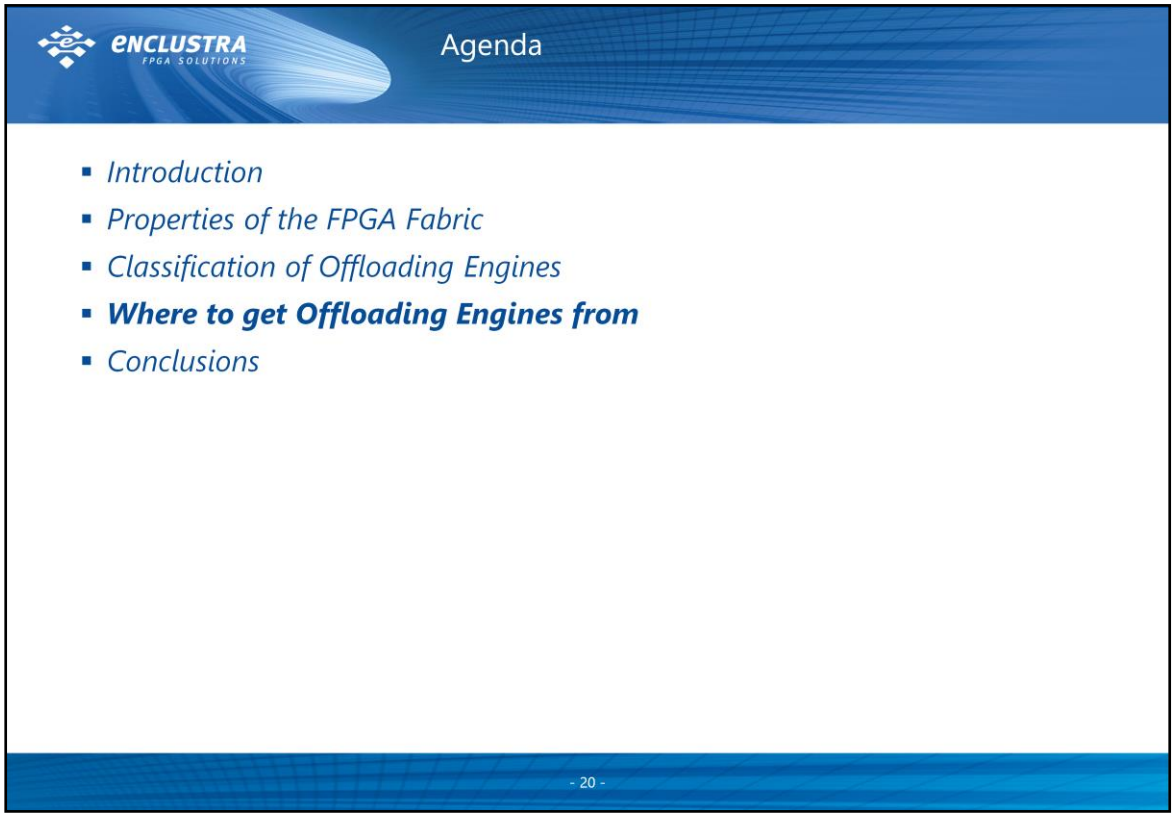


Example 2: Display Controller

This example shows a display controller. The controller contains a 2D accelerator to implement functions such as moving things on a display, displaying characters, drawing rectangles, etc. Of course the controller also regularly updates the actual display with new data.

The 2D accelerator nicely fits into the category «Software Accelerators» since it just changes the content of the video memory exactly the way the software could do, but it does it faster and without producing CPU load for just copying around some data. The display update is a good example for a «Output Processing» engine. It takes the data from the video memory and sends it to the display.

This example also shows the usage of direct access to a memory for large amounts of data (i.e. the frame to display) by the offloading engine. Interesting is that the CPU itself does not even need access to this memory because it is controlled by the offloading engine completely.



Where to get Offlaoding Engines from

You should now be familiar with offloading engines and their basic properties. But where do you get these engines from? Can you also profit from them without any chip design knowledge?

We will answer these questions with the following slides.

Enclustra GmbH
Räffelstrasse 28 – CH-8045 Zürich – Switzerland
Phone +41 43 343 39 43 – www.enclustra.com

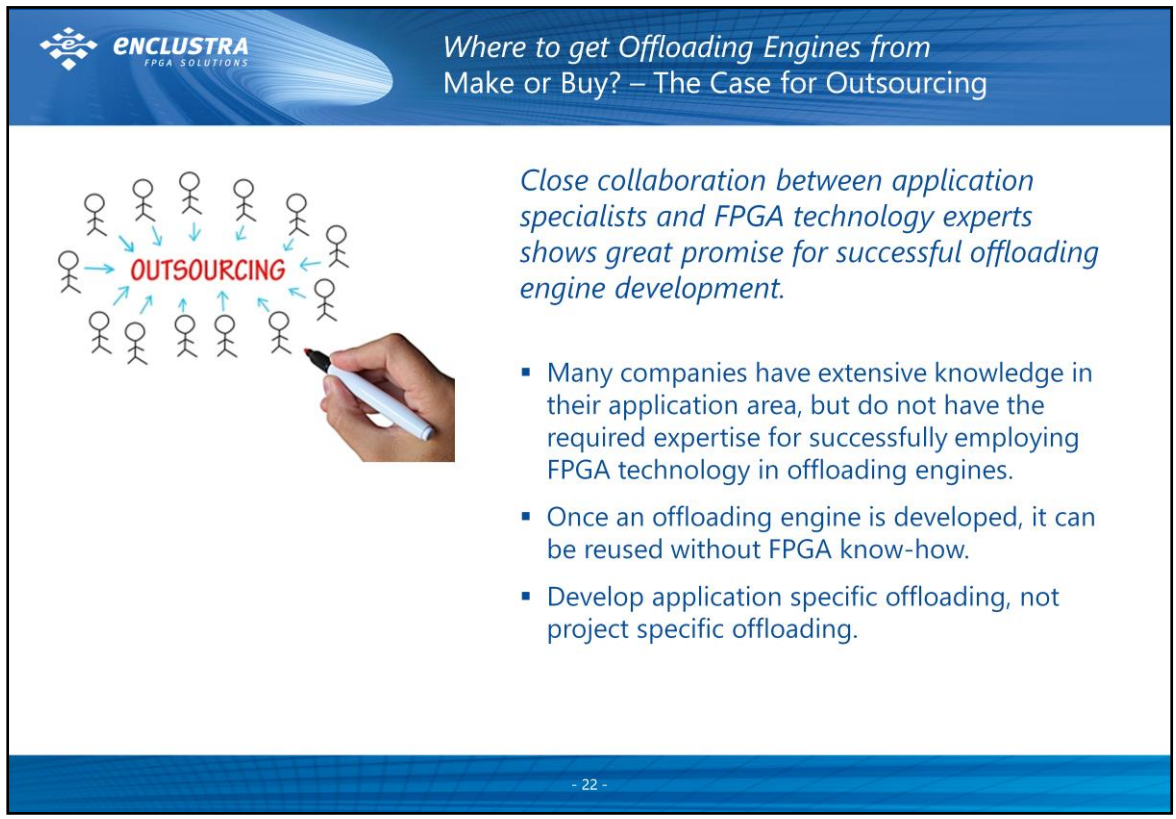
Where to get offloading engines from

There are many offloading engines available ready for integration without the need for developing them from scratch. While the NIH (Not Invented Here) syndrome still has some influence, these concerns usually get overruled by the fact that many systems today are too complex to be developed 100% in-house. Another important advantage is that an available offloading engine can be integrated into your SoC with no or very little chip design knowledge. These offloading engines are therefore accessible for embedded software developers without FPGA design experience as well.

Vendors of SoCs (e.g. Xilinx) and peripheral products (e.g. Analog Devices) offer many IP cores (offloading engines are a sub-category of IP cores) for free or at very little cost. Their main goal is to make their SoC devices attractive by reducing the effort for integrating them into your solution. Especially the IP cores delivered by peripheral product vendors often simply implement interfaces to the related product.

A completely different business model is the one of IP core vendors. These companies offer complex offloading engines for certain tasks but also with a significant price tag because they need to compensate the development effort. Nevertheless such IP cores are almost always cheaper than developing the same functionality from scratch.

If the functionality you need is very project specific or just not yet available on the market, offloading engines can be developed specifically for your project. Either by you or by a service company specialized in FPGA/SoC design, exactly like Enclustra.



Make or Buy? – The Case for Outsourcing

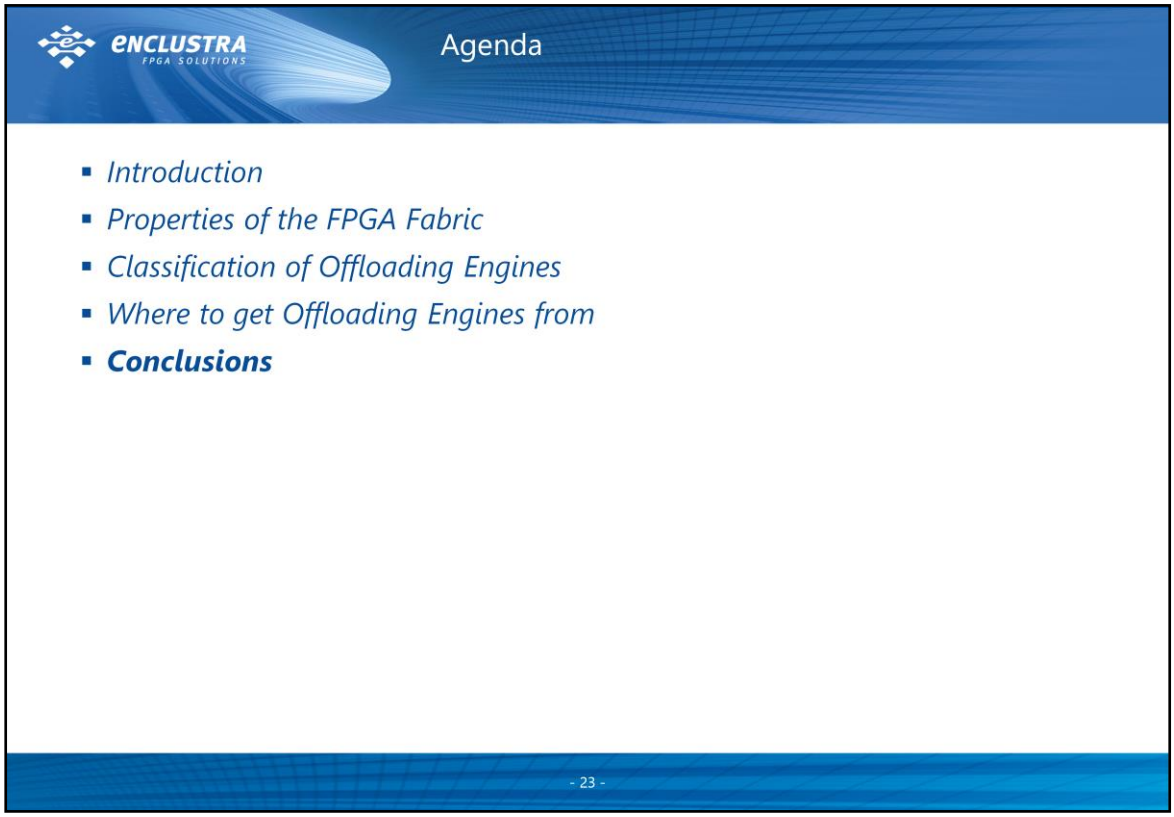
Successful and efficient development of offloading engines requires in-depth knowledge of

- Basic digital and analog circuit design, chip design, VLSI
- HDL (VHDL, Verilog, etc.)
- FPGA architecture and tools
- High-speed hardware design
- Deployed algorithms, I/O standards, protocols, etc.

Many companies have extensive knowledge in their application area, but do not have the required expertise for successfully employing FPGA/SoC technology. Furthermore, building up this know-how is a lengthy and expensive process.

Collaboration between application specialists and FPGA technology experts shows great promise for successful product development.

In many cases it is possible to develop offloading engines in a way they can be reused for in several projects. If an engine to be developed by a service company is specified "application specific" instead of "project specific" from beginning, the chances for being able to reuse it in future projects increase.


ENCLUSTRA
FPGA SOLUTIONS
Agenda
Introduction
Properties of the FPGA Fabric
Classification of Offloading Engines
Where to get Offloading Engines from
Conclusions
- 23 -

Conclusions

CPU offloading is omnipresent for Desktop PCs and going to conquer the domain of embedded processing.

- Offloading engines have the potential to multiply the computation power of a system.
- With SoCs project specific offloading engines become practical
- Each class of offloading engines has its own requirements.
- Most offloading engines do not fit exactly into one class – design decisions and trade-offs have to be taken.

- 24 -

Enclustra GmbH
Räffelstrasse 28 – CH-8045 Zürich – Switzerland
Phone +41 43 343 39 43 – www.enclustra.com

Questions?

Oliver Bründler
Project Leader
oliver.bruendler@enclustra.com
+41 43 343 39 48

Quarterly newsletter:
subscribe@enclustra.com

Upcoming Events:

- FPGA Kongress
  July 11-13, 2017
  Munich
  Germany

- 25 -

Image References

- *Slide 4 left image courtesy of Stoonn at FreeDigitalPhotos.net*
- *Slide 5 left image courtesy of vectorolie at FreeDigitalPhotos.net*
- *Slide 6 left image courtesy of jscreationzs and domeen at FreeDigitalPhotos.net*
- *Slide 21 left image courtesy of Stuart Miles at FreeDigitalPhotos.net*
- *Slide 22 left image courtesy of Stuart Miles at FreeDigitalPhotos.net*
- *Slide 24 left image courtesy of Vuono at FreeDigitalPhotos.net*
- *All other images courtesy of Enclustra GmbH*

Enclustra GmbH
Räffelstrasse 28 – CH-8045 Zürich – Switzerland
Phone +41 43 343 39 43 – www.enclustra.com