seeingmachines

The DMS Embedding Challenge

A Seeing Machines White Paper

Authors:
Timothy Edwards - Co-Founder, Strategic Advisor & former CTO
Rodney Stewart - Strategic Advisor & former Chief Engineer
Alif Wahid, PhD - Senior Staff Core Technology Architect

6 January 2022

Introduction

The automotive electronics market is characterised by high volume, high reliability and a supply chain that is in the ruthless pursuit of driving costs down. Any electronics that makes its way into a vehicle platform of even the most modest volumes, no matter the end user or the function, will have been custom designed, with a focus on cost optimization of the software and hardware components while still maximising performance and ensuring safety. This is the automotive embedding challenge.

High Performance Embedding is a technical discipline that Seeing Machines takes as seriously as algorithms or optics and we recognize it as the key to product scalability. After all, having industry-best algorithms and optical solutions means little if the ability to execute those algorithms comes at a prohibitive cost.

It was five to six years ago that camera based Driver Monitoring Systems (DMS) first entered into the common lexicon within automotive circles. By industry timelines, where programs typically take three or more years to execute, this is akin to the blink of an eye. With these timescales in mind, we highlight how suboptimal SoC products (that also take multiple years to develop) have been attempting to serve the DMS market with designs that have not benefited from the foresight and detailed knowledge of how DMS and (more recently) OMS (Occupant Monitoring Systems) solutions need to work.

In this paper, we reflect on some of the challenges that we have faced in the embedding discipline, and discuss our strategic approach (including our motivation and approach) to our Occula NPU design.

Table of Contents

Automotive DMS and OMS Overview

The design challenge in any embedded system within the context of the automotive market, including the monitoring of *drivers, occupants or complete cabins* (which we call throughout this paper as **DMS**), is multifaceted and complex.

The automotive industry arguably has the largest and most complex supply chains of any industry. As such, market forces demand that suppliers deliver continual improvements in performance while continually reducing prices and adhering to some of the most demanding environmental and safety requirements of any industry. Given the volumes at stake, cost pressures are extreme and ultimately the sales price becomes the primary driver of what features or functions are “affordable” in embedded systems. Put simply, every cent matters, and those solutions that can embed and process the needed functions with the required key performance indicators (KPIs) using less processing resources, and therefore lower cost, will always have a market advantage.

This cost-first industry dynamic isn’t a new phenomenon. Embedding software in automotive electronics has always entailed walking a tightrope when it comes to the processing resources available for the task at hand. Whilst the need for upgradability (reprogrammability) and emerging subscription services may disrupt and change this dynamic somewhat, there will always be a strong cost versus performance pressure associated with the act of embedding any function into a vehicle. Seeing Machines refers to this dynamic as “Just Good Enough Processing” (JGEP) and it underpins our philosophy and our approach to the embedding discipline. To achieve JGEP requires a deep understanding of the target silicon architecture - warts and all.

Before diving in further, we’ll define some key terms and categories in this section, which are used throughout the paper. By “embedded system” we mean a module that performs a carefully specified and dedicated function, and which contains a sophisticated computer system. The hardware and mechanics of an embedded system are almost always bespoke and must be customized to some extent to meet unique requirements of a vehicle design (e.g. housing enclosure, sensors, processors, volatile and non-volatile memories, actuators, power and thermal management, etc.). The software in an embedded system is carefully targeted at the underlying hardware so as to optimally implement the features required of the overall module.

An embedded “processing pipeline” refers to a decomposition of the overall algorithms into sequential stages. Achieving a smooth flow of data down the pipeline is the goal of embedded optimization. In its *simplest* form, the pipeline can be “software-only”, where the only optimisation work required may be no more than experimenting with the settings of the compiler, or perhaps writing some handcrafted assembly code to make particular inner loops of the program more efficient. In automotive embedded systems that contain DMS functionality (and indeed in all advanced computer-vision systems in cars), it is fair to say that there is almost always the need to accelerate stages of the pipeline using specialist processor designs. This isn’t because regular CPUs aren’t capable of executing the pipeline, but simply because the CPU resources made available are almost never sufficient (due to cost) and because there are usually far more efficient ways of executing specialized types of functions, including image pre-processing (e.g. denoise, dewarp), computer-vision operators (e.g. template matching, optical flows), signal processing (e.g. kalman filters), and neural networks such as regressors

and classifiers. The prevailing idea that any DMS supplier is able to offer highly compute-intensive product solutions in a software-only form, and be agnostic to the architectural differences in industry leading System-on-Chips (SoCs), whilst also being commercially competitive, is a fallacy.

Specific to DMS, the automotive industry is also grappling with other challenges that impact the embedding problem. In particular, the physical location of the optical sensing components (typically image sensors, lenses and illumination pods) along with other factors such as new regulatory demands imposed in Europe and other jurisdictions. All of these factors influence and skew a vehicle OEM's typical "technology introduction" playbook, where emerging technologies (such as DMS) are carefully staged and controlled by the OEM in order to manage risk.

OEMs typically introduce new technologies as "**standalone**" systems (as in a single-purpose system in a box with its own dedicated ECU). The technology and supply chain is matured and optimized first; then in later generations the OEMs begin to push for "**integration**" with other systems. This stage is driven by the insatiable desire to optimize cost, power, weight and other variables that fundamentally affect the making of a car.

These are not new phenomena - the age-old question of Integrated versus Standalone for a given function is actually a swinging pendulum. Some OEMs push hard towards integration as quickly as possible. Others wait for the industry to commoditise the technology and iron out integration issues (and costs). The picture in **Figure 1** highlights the main landing points of DMS systems (optics and/or processing) within a vehicle.

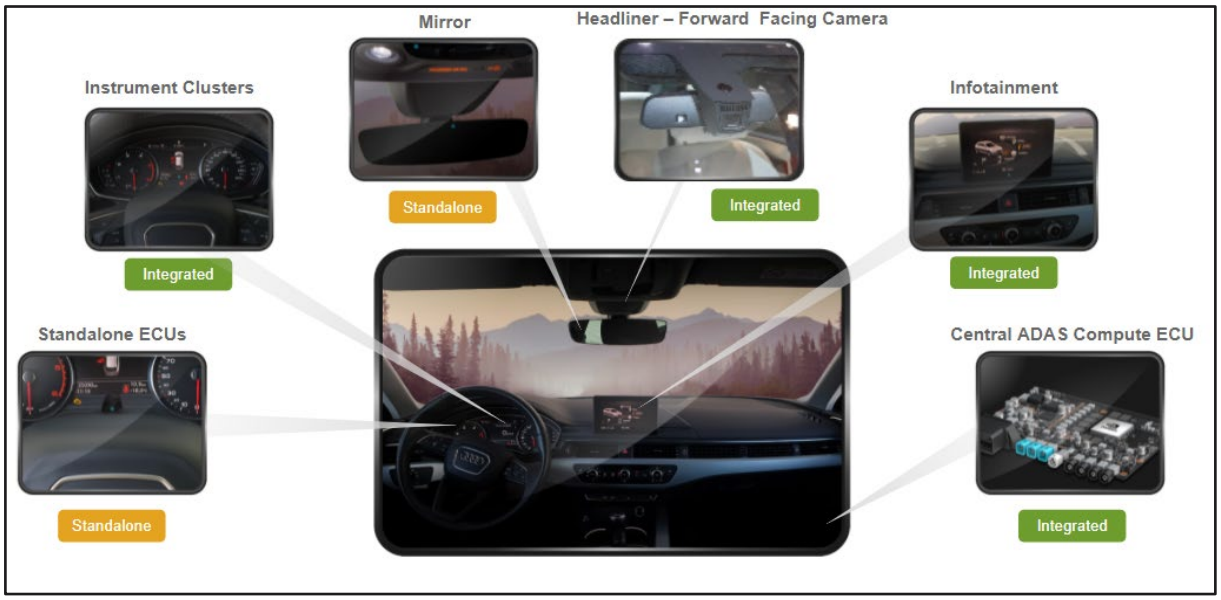


Figure 1: Overview of landing points for DMS Processors and Optics

Where exactly in a vehicle DMS ultimately ends up landing (including the processing component) will continue to play out over this decade. What appears certain, however, is that there will be a healthy balance between standalone and integrated solutions during this time

(perhaps equally split between the two), and not all OEMs will pursue the same strategy when deciding which path to take for their vehicle lines.

It is also important to keep in mind that bodies like the Euro NCAP (New Car Assessment Program) and other regulatory groups around the world are now requiring DMS technology incorporation into new vehicles as soon as 2023. This is a regulatory trajectory that appears akin to other safety systems ranging from airbags and seatbelts to select advanced driver assistance systems (ADAS). The complexity of DMS combined with the regulatory timing demand, places immense pressure on those OEMs that have not completed their typical first phase of understanding and commoditization. These companies will be forced to adopt a “fast follower” approach, duplicating systems that “tick the box” of DMS regulatory requirements. This is creating a wave of OEM demand for standalone DMS solutions due to the simpler engineering and reduced timing risk when needing to incorporate DMS into a large number of vehicle lines. Given this market dynamic, the volume of modules expected and the “automotive jungle” where all possible solutions compete for survival - SoCs that target DMS standalone solutions are fast becoming an absolute necessity.

Standalone Systems: The Embedded Battleground

DMS products are inherently high performance, real-time computing systems. From a processing perspective, video is pushed into a pipeline which operates on the continuous stream of pixels, in a series of stages, where hierarchies of interconnected algorithms (or a DMS "engine") squeeze out and extract desired information from the image data, passing higher-order information from one stage to the next, until what is produced is a low-bandwidth set of high-level (and high-value) results, such as where the driver is looking and if they are distracted or impaired. These results are then transmitted out of the DMS module into the vehicle environment for consumption by any number of downstream modules (e.g. a wider system of ADAS modules or some form of automated driving system).

Whilst the engineering methods applied when embedding a DMS engine into any target SoC follow the same set of general steps, the risks can be very different based on the specifics of the processor. As a general rule, processors used in standalone systems are lower cost, smaller and far less capable than those in integrated systems. Standalone processors are considered a far riskier embedded target because of "hard limits" commonly encountered with these types of processors. The graph in **Figure 2** below shows the theoretical maximum CPU and memory bandwidth capabilities of five SoCs: four of which are targeted at the standalone use-case and the fifth being targeted at integrated solutions. Even though there is a wide spectrum of processors available, those in the integrated class simply have far more compute resources as they are required to do a lot more than execute DMS workloads.

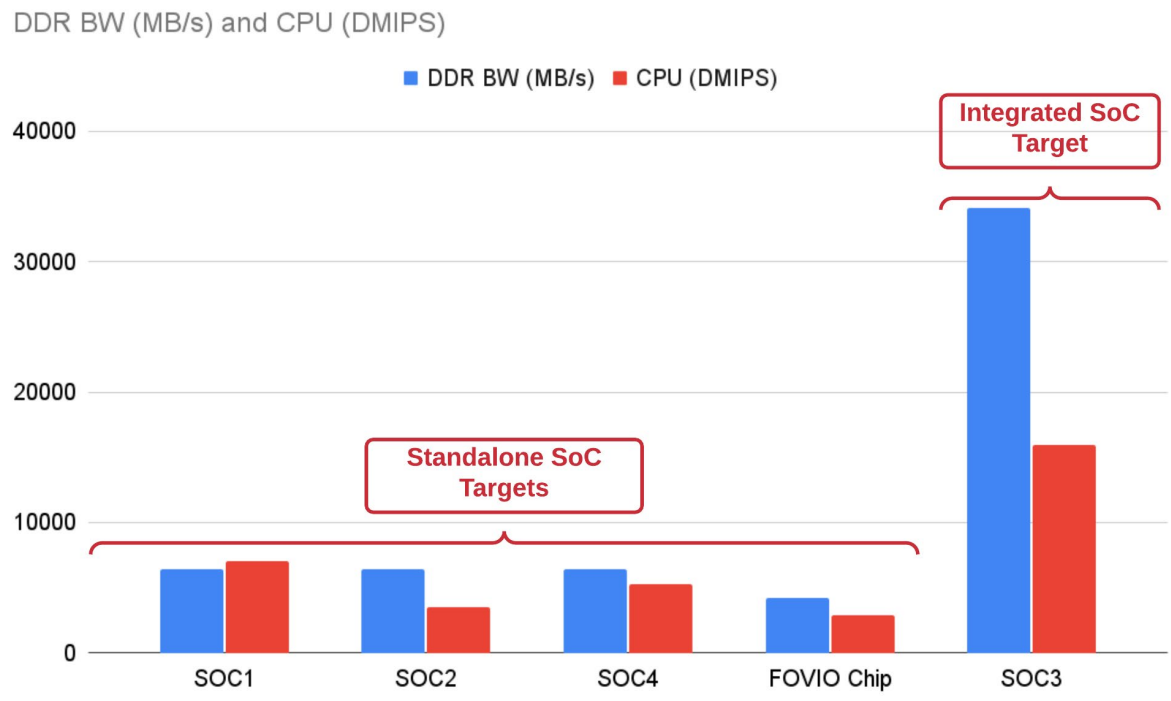


Figure 2: Standalone versus Integrated SoC processing resources

The reason that embedding into a standalone targeted processor is riskier is quite simple - SOCs targeted at integrated solutions have many more "design levers" and far more CPU cores, co-processors and memory bandwidth than those targeted at standalone solutions. In our experience, they can be anywhere from five to twenty times more powerful than what is

required to implement a real-time DMS system. Hence, if there are issues encountered when embedding the DMS function into an integrated solution, then there is almost always a way to “shuffle” around execution aspects in order to better balance resource demands and free critical compute resources for the pipeline. In contrast, standalone systems have *just enough* resources to do a single job - rarely more.

For standalone systems, in the event that a pipeline cannot be mapped to the silicon efficiently (which in our experience is very common), there is really no other option but to move to a bigger, and therefore more expensive processor (or for the algorithm's processing rate, and consequent performance, to not only suffer but to suffer unpredictably). More often than not, it is impossible to move to another processor because it's either discovered too late in a project or it's simply not commercially viable. This situation can easily end with all parties running out of time and the DMS software team being forced to find non-existent ways of fitting the processing into the SoC, and the OEM integration team ultimately being severely disappointed while having to scale back their expectations for any features that depend on the DMS output results.

This dynamic places a lot of pressure on the embedding process and creates a high stakes game when *estimating and mapping* pipelines to target chips confidently in *advance*. The starting point, from our perspective, is to ensure that we are working with chips that are compatible with our pipelines, as we know from experience that not all chips are built equally (even if the specifications on paper look indistinguishable). This requires a deep understanding of our algorithms, our dataflows and the target chip architectures.

DMS-Oriented Chip Architecture

Today, the DMS embedded processing market is at a fork in the road. Up until very recently, the chips that have been targeted at DMS never anticipated an optimal processing pipeline or the specific operations that a DMS workload needs. Instead, the devices being offered for standalone DMS processing appear to be repurposed devices originally targeted at outward facing vision processing pipelines and are sub-optimally matched to the nature of processing required to search and track *human bodies* (particularly faces, eyes, hands and torsos).

Our view is that DMS-oriented chips require the following architectural traits:

(i) Sufficiently powerful CPU cores (usually two or more) with vector extensions and shared L2 cache (typically 256 KB or larger),

(ii) Optimised video capture and ISP pipeline that never-ever tax the system memory bandwidth,

(iii) Well matched accelerator(s) to the algorithmic tasks at hand,

(iv) Essential I/O peripherals for connecting to vehicle interfaces as well as controlling functions like illumination, and

(v) Functional safety plus cybersecurity built in from the ground up for all relevant subsystems (e.g. secure bootflow, biometric data protection and comprehensive soft-error mitigation).

Current devices in the market that are targeted at standalone DMS applications do not possess *all* of the aforementioned traits. There is typically a shortfall in at least one of these traits that make it very challenging to develop a superior DMS solution. Many existing devices lack anticipated advances in image sensor technologies - particularly RGB-IR imagers. The typical comment by most chip makers to the issue of RGB-IR sensors is that "*Our Image Signal Processor (ISP) is capable of handling all necessary RGB-IR image operations*". While that is a technically defensible statement, when we study the problem in the next section we will illustrate that there are many pitfalls around RGB-IR sensors which can easily trap DMS product developers.

Memory Bandwidth - A Precious Resource

Semiconductor companies and the powerful marketing machinery at their disposal will try to have engineers, architects and other users of their chips believe that it's all about the number of DMIPS that the CPUs achieve, the number of TOPS that the NPU (or perhaps a DSP accelerator) is capable of, and the feature-rich capabilities of the "all singing and dancing" ISP. But what about concurrency? It all sounds great until you start using these SoCs and quickly realize that while the lowest cost devices can do what the vendor says, they can only do one feature at a time due to overall SOC architectural limitations. "Real world" concurrent feature requirements quickly become unaffordable as a result of being forced into devices with higher processing performance. These performance issues, especially in the smaller chips (i.e. the devices used in standalone DMS applications), are often not due to the primary resources such as the CPU or an accelerator, but rather the memory subsystem being saturated with the compute resources being starved of data when attempting to run many features concurrently. Whether or not it's running out of processing cycles or memory bandwidth, the end result is the same - the real-time pipeline stalls and misses the targeted deadline for delivering DMS results.

For memory specifications, chipmakers usually state the external device type and speed in their datasheets, and often the raw theoretical (but unachievable) maximum bandwidth numbers and leave it at that. Memory is the neglected, unsexy commodity resource that doesn't allow for any product differentiation, and therefore is rarely focused on early in the device selection process. The reality of embedded systems is that crucial secondary resources such as memory bandwidth can often be more important than the total throughput of the CPU or the capabilities of any accelerator.

Most important, however, are the subtle details of the chip architecture with all the potential ways that data can be moved around the chip and possibly buffered along the way with internal on-chip memory. The art of efficient embedding on a given chip for a processing pipeline in many ways is about minimizing the need to move data around and access the same data more than once. Equally, the art of good chip design is to make sure that the processing data-paths are clearly defined so that the chip enables the developer to minimise the use of critical resources such as external memory. This dynamic looms large on the horizon as chips become smaller and more resource constrained, since memory is limited both on and off the chip. Therefore inefficient pipelines and chip bottlenecks get exposed very quickly.

Case-Study - RGB-IR Preprocessing

RGB-IR imagers are new to the automotive market and stem from the familiar OEM desire to optimize and reduce cost. Which is to say, *why have two image sensors when you can have one*?

RGB-IR imagers allow one sensor to create two separate video streams in both visible-light (RGB) and narrowband infrared (IR). The RGB stream supports functions like web conferencing, selfies, security or anything that the human eye might consume. The second stream (IR) is consumed by machine vision for the traditional DMS safety functions, where the lighting is designed to be both immune to sunlight pollution as well as to see in the dark.

Putting aside all of the complex optical challenges that RGB-IR imagers create, they also present new challenges in the embedding space: particularly on the Video Input and Preprocessing stages of the pipeline. Put simply, most chips targeted at the standalone DMS market have not been designed to **efficiently** allow for the capture, demux, and processing of both the RGB and IR video streams of interest from an RGB-IR sensor. This is a major drawback and reflective of the fact that most chips, when originally architected, didn't anticipate the use of these sensor types, nor how the DMS algorithms might consume them.

To illustrate this issue and to show why this is a very important consideration when selecting an SoC, let's consider both streams individually and focus on the Video Input and Preprocessing stage of the pipeline.

An ideal IR-only stream is typically very straightforward - machine vision wants to see and use an unadulterated raw video stream. An Ideal pipeline shown in Figure 3 will implement a Downscale Pyramid where different image scales are created on the fly. Dewarping is typically not needed in this part of the processing pipeline. There may be minor filtering needed in the front-end Video Input block depending on the sensor, but nothing complex.

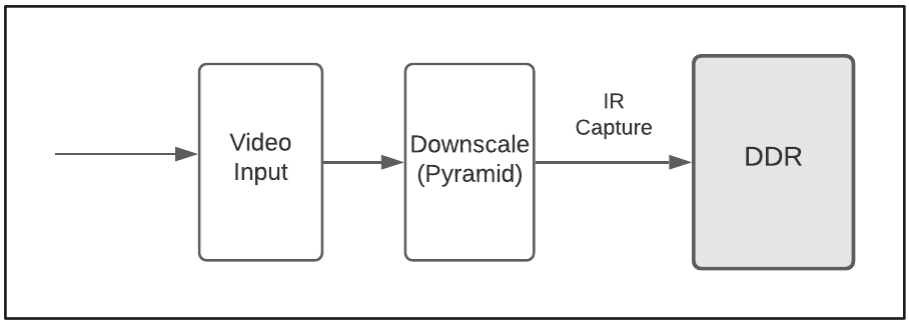


Figure 3: Example of Ideal IR-only Video pipeline

An RGB stream is more complex as the pixels need to be conditioned for consumption by the human eye, and therefore functions like lens dewarping, sharpness enhancement, colour correction, masking and background blurring for privacy protection may also be required. An example Ideal RGB Video Input and Preprocessing Stage may look like Figure 4.

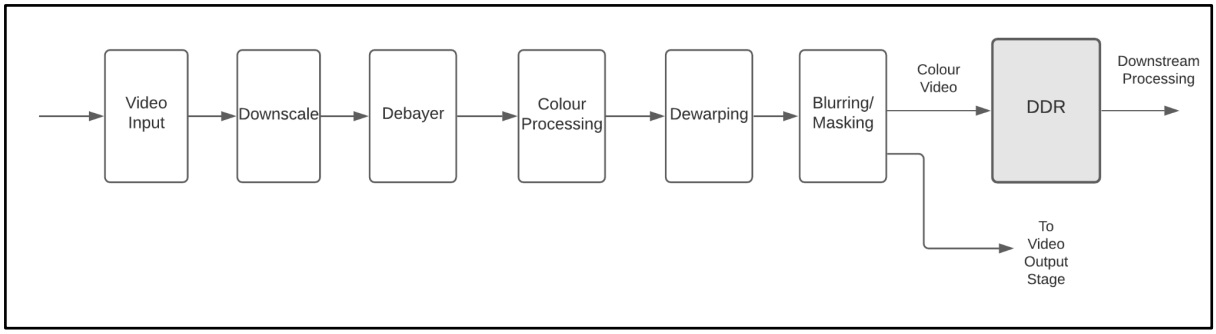


Figure 4: Example of Ideal RGB-only Video pipeline

This is where things get interesting, and possibly more difficult for SoC vendors, especially when they do not know the dataflows required for the end application. Given that they are also influenced heavily by the need to build chips that service a far wider application space than say DMS and OMS, they tend to allow for configurable data-paths and functions - thus enter the programmable ISP.

The ideal architecture shown in **Figure 4** allows for maximum efficiency, but minimal flexibility as the dataflow is fixed. This architecture will perform all of the processing required on the fly (exploiting small on-chip *line buffers* where essential) and send the resulting image(s) into DDR memory at the end of the pipeline. This type of architecture minimizes the traffic on the DDR memory (a good thing), but as a tradeoff it can impose restrictions on the functions and the data-paths (maybe not so good for the SoC vendor who is trying to target other use-cases).

At the other end of the architecture spectrum is the “shared memory” design. Using those same functional blocks above, the pipeline might look something like the diagram below in **Figure 5**. Note that although chip datasheets would typically put one big box here and call it an “ISP”, we have kept it as a functional diagram to highlight the memory transactions (trips) that are still necessary to achieve the processing required of the pipeline.

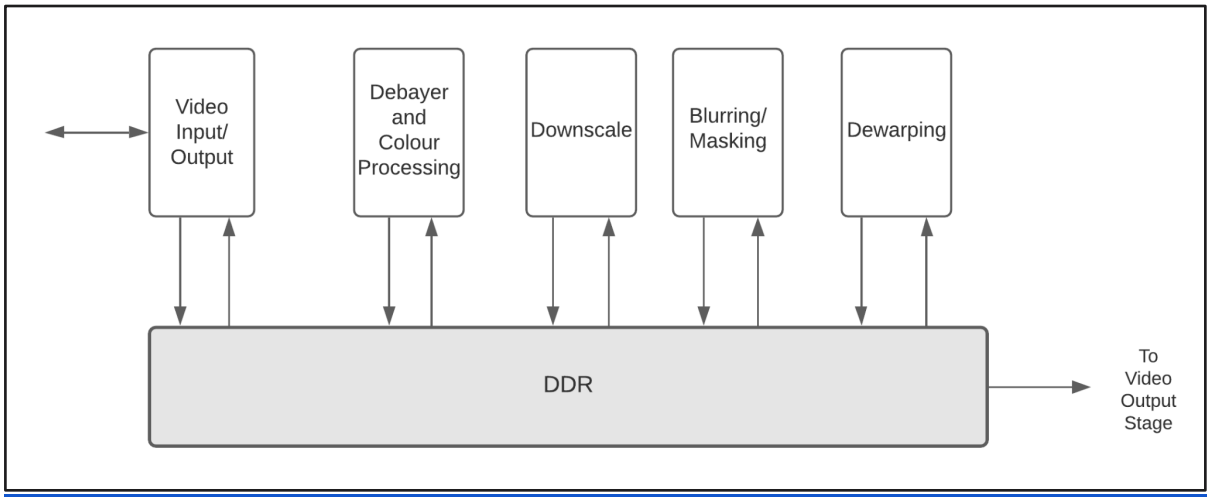


Figure 5: Example of Alternative RGB-only Video Input Stage using Shared Memory

The shared memory architecture allows maximum flexibility but comes at the cost of external memory bandwidth efficiency. The pipeline basically reads and writes images (or parts of images) in and out of external memory at every stage. It allows for almost any combination of

datapath, and if the processing elements are “programmable” then maximum flexibility is available. For larger SoC’s that have plenty of memory bandwidth, it is an easy decision to take this sort of design approach. For SoC’s and systems that are limited on memory bandwidth, for example small standalone DMS processors, an architecture that relies purely on shared memory connectivity for its ISP can be seriously problematic (even to the point of making chips that appear powerful on paper simply unusable for DMS solutions). It is very easy to consume memory bandwidth getting video in and out of the device and to “starve” the rest of the device, as we will demonstrate.

Through many programs and RFQs over the years at Seeing Machines, we have had the good fortune of mapping our algorithms and solutions to many different SoCs and we are highly tuned into this problem along with many other chip architecture related issues that present themselves. Nearly all of the issues that we see when mapping to a given chip end up being associated with inefficiencies around dataflow and the suboptimal use of both on-chip and off-chip memory. We have found that accelerators will often achieve lightning fast numerical operations only to let us down badly in the act of moving the data to the next stage of the pipeline and being forced to buffer data in external DDR memory. This is again symptomatic of the chip simply not being designed with the DMS-oriented pipeline configuration in mind.

Now let’s turn these ideas into numbers and illustrate the point clearly. Table 1 highlights the range of memory bandwidths required to perform the same RGB-IR video input function on a range of SoCs targeted at the standalone DMS market. We’re using a 5MP sensor running at 60Hz for this scenario, where half the frames are IR and the other half are RGB in an alternating fashion. This table and its adjacent plot visually highlight the enormous variance that exists purely due to the different video pipelines we have outlined above. Note how certain SoCs can be **at least 6x worse** when compared to the Ideal pipelines we’ve articulated.

SOC	DDR R/W Trips	DDR R/W Bandwidth (MB/s)
IDEAL	1	290.0
SOC1	6	1800.0
SOC2	6	2400.0
SOC3	8	2500.0
SOC4	4	620.0
FOVIO Chip	1	320.0

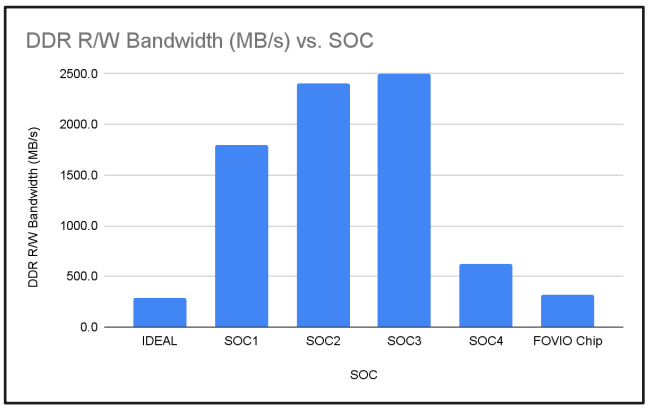


Table 1: Comparison of DDR B/W used for the same RGB-IR function (lower is better)

So why does this matter? The answer is quite simple - we haven’t even begun the primary job required of the chip, which is the vision processing task, let alone fulfill other requirements such as video output (which itself can also be demanding of memory bandwidth).

This may not always be a problem depending on the chip in question and requirements of the DMS module (e.g. IR-only monochrome sensor without any ISP nor any H.264 encoder is obviously less demanding of memory bandwidth). What ultimately matters is whether the device has enough memory bandwidth to confidently execute the vision processing job after subtracting the video input and output allocations.This question is not straightforward to answer by any means, as there are many ways to solve the same problem and the chip architecture's "suitability" when mapping the desired vision processing pipeline.

Moving and pre-processing video in and out of a chip is generally deterministic with respect to the memory bandwidth requirements. Typically, it is easy to calculate and therefore allocate resources. The only exception is compression on the video output stage which can vary based on the pixel content. This is a load that needs to be pre-allocated before determining what is then available for the vision processing for any given chipset. A simple view of the main functional stages of a DMS processing pipeline is shown in **Figure 6** using an example 5MP - 60Hz - RGB-IR sensor and Video Output with local H.264 compression at 30Hz for Full HD resolution of 1920 x 1080.

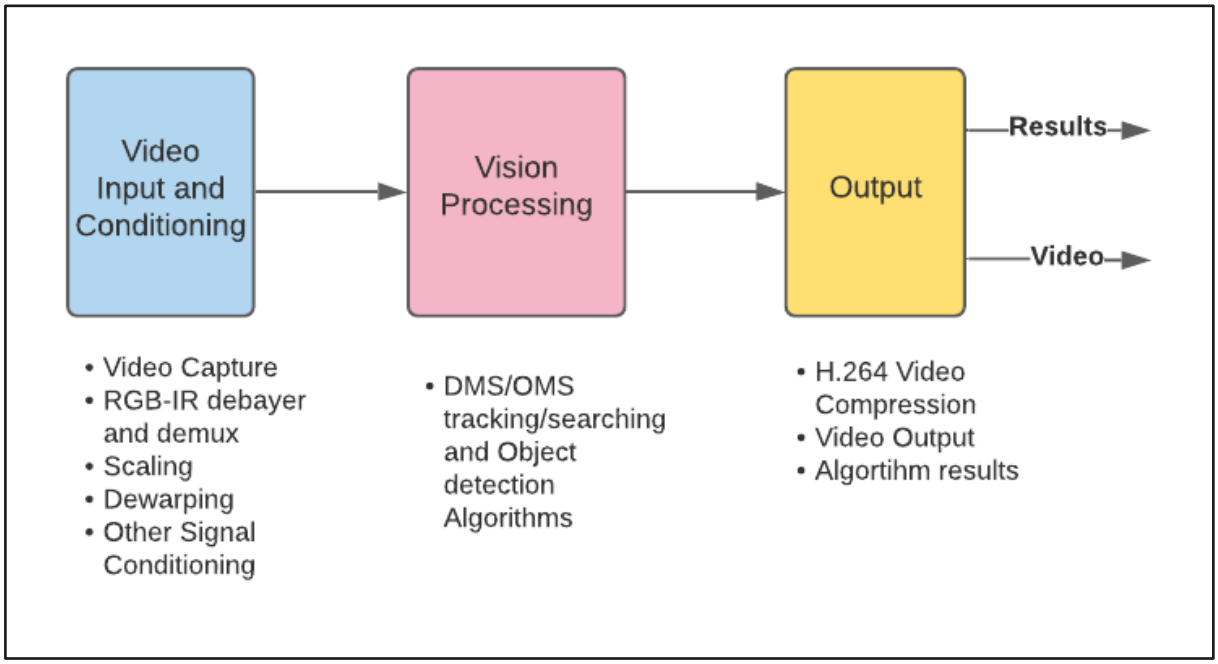


Figure 6: Main stages of a DMS processing Pipeline

Table 2 below translates this into numbers and highlights the importance of looking past the marketing propaganda on any particular chip resource. For a DMS product, what actually matters is the resources remaining after all other tasks (and reserved allocations) have been accounted for (which is the column on the right, highlighted in pink).

				Allocated BW		Total Bandwidth		
	Imager	Compression	DDR Type	Video Input BW inc. RGB-IR capture (MB/s)	Video Output BW inc. Compression (MB/s)	Theoretical Max (MB/s)	DDR Efficiency @ 60% (MB/s)	Remaining BW (MB/s)
SOC1	5MP RGB-IR @60 Hz	H.264 1080p p-Frame	16 bit LPDDR4 @1600MHz	1800	350	6400	3840	**1690**
SOC2	5MP RGB-IR @60 Hz	H.264 1080p p-Frame	16 bit LPDDR4 @1600MHz	2400	350	6400	3840	**1090**
SOC3	5MP RGB-IR @60 Hz	H.264 1080p p-Frame	64 bit LPDDR4 @2133MHz	2500	350	34128	20477	**17627**
SOC4	5MP RGB-IR @60 Hz	H.264 1080p p-Frame	32 bit DDR3L @800MHz	620	350	6400	3840	**2870**
FOVIO Chip	5MP RGB-IR @60 Hz	H.264 1080p p-Frame	32 bit DDR3L @533MHz	320	550	4264	2558	**1688**

Table 2: Overview of the residual bandwidth available for Vision Processing

It is interesting to note that some SoCs might have large total memory bandwidth, but it doesn't always equate to *usable bandwidth* for the DMS workload. These numbers show that there are two SoC's in Table 2 where **more** DDR bandwidth *must be* assigned to the Video Input and Output stages for a system using a 5MP RGB-IR imager than what can be made available to the Vision Processing task, which is the most important job of the processor in this application! This is clearly not acceptable and generally rules those SoCs out, or at least severely limits their usability. The poor efficiency of the RGB-IR processing essentially means that any advantage those devices might claim on DDR bandwidth is an illusion.

Another view of the data summarised in the table above is the following graph of Figure 7. When analysing the suitability of any SoC for a DMS system, the primary parameter of importance is the yellow section on each bar which represents the *real bandwidth* remaining that can be allocated to DMS algorithm processing. The absolute amount of DDR bandwidth remaining, not the percentage of total bandwidth, is all that matters in this view. The bigger the yellow regions, the smaller the chances of DDR bandwidth creating bottlenecks in the overall processing pipeline.

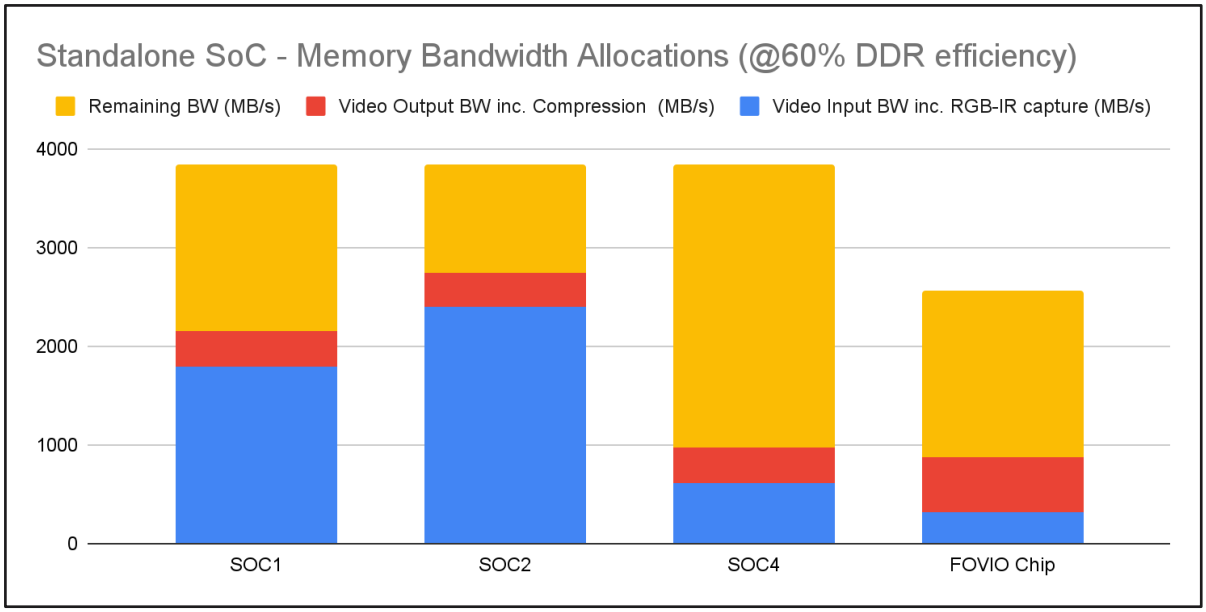


Figure 7: Comparison of some Standalone DMS SoCs' bandwidth allocations with RGB-IR and H.264 processing

It must be noted that these comparisons are for a particular subset of SoCs offered in the market. There are bigger (and smaller) devices in different product families that offer wider (and narrower) memory interfaces. The specific SoCs, compared here, generally compete commercially in the same space and have been selected to highlight the effect of silicon architecture inefficiencies on low-end devices for a task they were not necessarily designed to efficiently perform.

This case-study is a really good example of what can happen to an important chip resource if the silicon isn't matched to the processing task. On the flipside, our analysis shows that SOC4 has clearly been designed to match the processing task of the RGB-IR extraction with the bandwidth close to that of an Ideal pipeline. This has been achieved by the vendor whilst still allowing for significant flexibility in the ISP and providing a large amount of residual bandwidth for DMS workload.

Embedded Accelerators and Neural Networks

Introduction to Neural Networks

The field of machine learning, nowadays also known as deep learning, has undergone a step-change in performance for many classes of algorithms - especially those in computer vision. The term “deep learning” ultimately encompasses techniques for the efficient discovery and encoding of relationships present in data into neural network (NN) “models”. These models are mathematical constructs consisting of a set of nodes (or artificial neurons) and connections between the nodes (or artificial synapses). Each node implements a simple mathematical function (or “activation” function) to be applied to the inputs and passed via the outputs to the next node. Networks are usually structured as layers of nodes that share a common activation function. The layered approach lends itself to implementations where layers are processed as discrete stages in a processing pipeline, from input layer, through “hidden” layers, to the final output layer.

Deep-learned networks are “deep” in the sense that there may be dozens of hidden layers. More exotic network architectures allow connections beyond just adjacent layers or which may have recurrent (or cyclic) connections that allow for feedback loops and stateful memory effects.

The design dimensions of a network include: (i) the type of activation function in each layer, (ii) scaling and threshold coefficients applied to each node’s inputs and activation function, (iii) the overall node and layer connectivity (or topology), (iv) the number of nodes per layer and number of layers overall. These design dimensions are the “free variables” in the algorithm development and training process. The magic of deep learning technology lies in training algorithms that are able to search the vast space of all the combinations of these free variables and with only a little data, quickly converge towards network designs which incorporate the relationships in the data into the NN model. Consequently, it is often the “training environment” that is a key ingredient in state-of-the-art deep learning technology, and considered valuable IP for any technology company.

Today there are many well known “classes” of networks. These are network topologies that have been researched to out-perform other legacy classes (when paired with specialized training environments for that class). Network classes are assigned names which become well known in the machine learning community, (e.g. “LSTM”, “MobileNet”, “ResNet”, and many others).

Over the last decade network classes have been the subject of intense research and have evolved quickly. This evolution is a result of the pressure for networks to (i) be trained in fewer compute cycles, using less or lower-quality training data, (ii) deliver superior performance outcomes to existing classes, whilst (iii) producing networks that are smaller and faster to execute, and therefore reduce the embedded processing costs. This forms a critical part of the Seeing Machines JGEP optimisation cycle.

While deep learning and NNs have a long history in academic circles, they began to enter commercial products around a decade ago. This was the beginning of the classic technology disruption “S-curve” and it commenced in the fields of natural language understanding and

image classification for internet search engines. Custom silicon NPU acceleration for data-centers (cloud) soon followed, including designs evolved from GPUs primarily by NVIDIA, custom ASIC solutions from Google, and FPGA implementations by Microsoft. These helped drive the initial mass adoption of “deep-learned AI” into many everyday (online) products.

Today, deep-learned AI has largely displaced even the best classical techniques in the field of computer vision. “AI” algorithms are powering many new types of products, not only in the cloud, but at the “edge”, running locally on smartphones and some IoT devices. Toolkits for efficient development of NNs (such as TensorFlow, Torch, ONNX and others) are now highly mature and have given rise to an enormous developer community, and thereby a whole new class of products often referred to as “AI powered”.

While these products are not really intelligent (they just detect patterns very well), they perhaps earn the “AI” moniker by being able to out-perform humans for the task of speedily extracting meaningful and actionable information from complex and noisy data. However, most networks remain highly specialized and only perform “narrow” tasks, such as face recognition. As such, they are more accurately termed “narrow AI” or to those that are old school, just very good “expert systems”.

Cost of Processing Challenge

Despite all their aforementioned advantages in the algorithmic space, however, NNs do not come without drawbacks - the most significant being a high processing demand when compared to classical techniques.

We define Cost of Processing (COP) as that part of a product’s cost structure which is driven directly or indirectly by algorithm processing demands. COP can be a difficult value to estimate without actually building competing solutions, as it consists not only of the cost of additional silicon dedicated to performing the number-crunching, but also the dependent costs of things like additional PCB space, thermal and power management aspects such as capacitors, inductors, power-management ICs, additional board-layers, more pins, larger heatsinks, outer packaging materials, and any additional IP licenses (such as third-party accelerator designs).

COP is a major barrier that limits the size of NNs, their performance and ultimately the proliferation of NNs into low-power, low-cost products. This barrier is due to the fact that (i) NNs have data-flows and operators that are unable to be efficiently executed on traditional Von-Neumann CPU designs such as ARM or Intel x86 cores, and (ii) generic NPUs are not necessarily a silver-bullet, potentially adding their own cost and power demands to any SoC.

Memory Bandwidth and Neural Networks

Now let’s put the previous chapter’s memory bandwidth numbers into context as one of the principal dimensions of the Cost of Processing. We’ll consider some common NNs used in embedded systems and understand what it takes from a memory bandwidth perspective to run a quantized model (8-bits) at a given processing rate. Note that we will explore NNs in more detail in the next section regarding the acceleration of their execution.

		Raw Memory BW (MB/s) - No Caching, 224x224			
	Data Transfered (MB)	1Hz	5Hz	10Hz	30Hz
MobileNetV3 Large	31.4	31.4	157.0	314.0	942.0
MobileNetV3 Small	11.1	11.1	55.3	110.7	332.0
Resnet-50	135.7	135.7	678.5	1357.0	4070.9

Table 3: Raw Memory Bandwidth requirements of common Neural Networks for different processing rates without local/internal cache, while using INT8 quantized models.

Table 3 represents the worst-case scenario as it assumes an accelerator that has no local cache which can be called upon to relieve the DDR memory accesses. These numbers can be reduced a lot, sometimes by at least 50% or more, *if local caches can be smartly utilized*. This is certainly not always the case; in fact, such caches are sometimes not usable unless the entire network can be executed out of the cache. Quite often, given the nature of the real-time processing in a DMS at 30fps (or sometimes higher), the networks need to be scheduled carefully in order to satisfy the 33 ms periodic deadline for DMS results (or even shorter periodic deadline when the processing rate is higher). This severely limits the possibility of "batching" network runs in contiguous blocks, for which many of the available accelerators have been optimised. Another way to appreciate this is the old adage that "*throughput can be bought, but latency must be earned*". Accelerators are designed to maximize throughput, but in the process of doing so they can elongate the latency of real-time tasks and often fail to hit deadlines.

Putting aside improvements to the DDR bandwidth from onboard caching, it can be easily seen that to run any of these networks at higher processing rates quickly starts to require significant memory bandwidth. Whilst Seeing Machines algorithms are not purely based on the networks outlined in this table, it is useful to show how easily processing at these higher rates can rapidly consume the resource budgets of devices in this class (it is why the Occula NPU and the custom networks we run on Occula were invented). There are plenty of networks (most actually) that do not need to run anywhere near the rates required with "tracking" algorithms. For example, some DMS designs have required **ten or more** networks to be implemented that execute concurrently at vastly different rates.

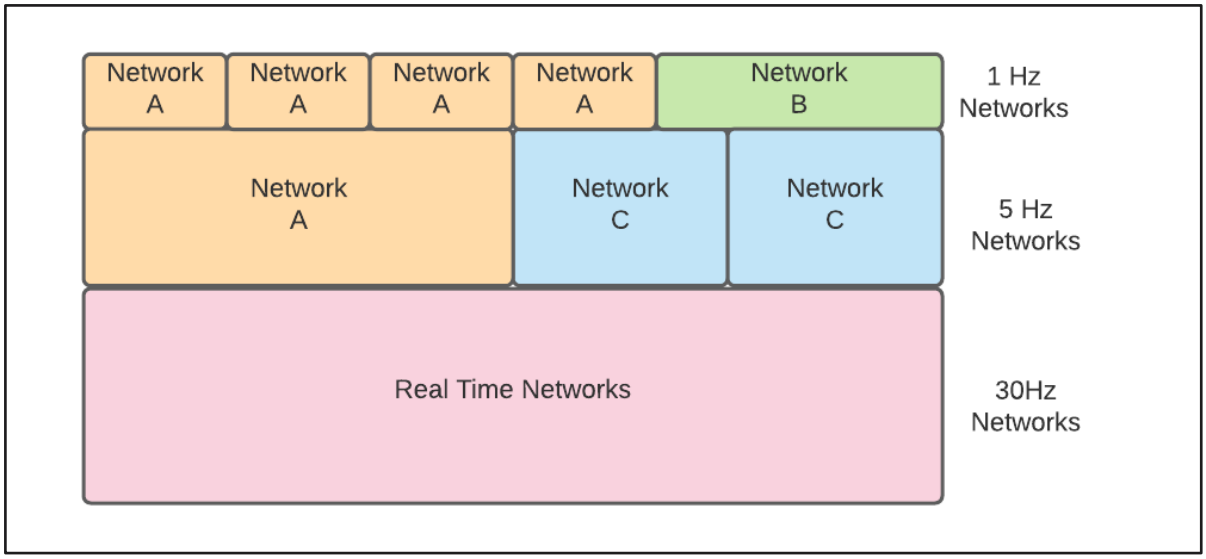


Figure 8: Relative size of bandwidth allocated to various networks at different processing rates

The diagram shown in **Figure 8** is a visual example where the size of each box represents the relative allocation of the available memory bandwidth to the different networks (separate colours) and their required processing rates (separate layers). The type of neural network for any given task is out of scope for this paper other than to say that different NN's have varying performance characteristics and are therefore directly related to the KPIs of the required features. The algorithm requirements and performance will dictate the type of network and the minimum processing rates needed. The processing pipeline must ensure that each network is executed at no less than its required processing rate and is always able to satisfy its periodic deadline (i.e. batching is not an automatic option).

Embedded NPUs

The awkward fit of NNs to execute on conventional CPUs combined with exponentially rising fabrication costs at smaller transistor design nodes (in what appears to be the end of the Moore's Law[1] in the cost-sensitive automotive world for the foreseeable future), has led to a Cambrian explosion[2] of Neural Processing Unit (NPU) accelerator designs over the past few years.

These new NPU co-processors have transistor groupings and memory arrangements that attempt to optimize the execution for whatever classes of NNs the chip designers believe will be popular with algorithm developers. Different designs vary due to the specific niche being sought for the product. Some NPUs are sold as standalone co-processing chips, others are licensable cores. Some employ new silicon techniques such as in-memory computing using nano-scale resistors.

[1] https://www.technologyreview.com/2020/02/24/905789/were-not-prepared-for-the-end-of-moores-law/

[2] https://en.wikipedia.org/wiki/Cambrian_explosion

Embedded NPU designs nowadays differ in the number and variety of mathematical operators, the precision of those operators, the way blobs of data are arranged, fetched, cached and propagated when the network executes, and the maximum size of the network that can be locally processed inside the NPU core without requiring a DDR memory access or a significant data transfer. Some designs also incorporate multiple processing pipelines in order to increase parallelism and throughput (which comes at the cost of extra silicon area of course).

An Embedded NPU is simply one that is designed with the intent of executing NNs in low-cost, small, low-power devices “at the edge”. The constraints of low cost, size and power place significant limits on the NPU capability compared to designs for cloud data-centers, or workstations, and thus Embedded NPUs must make compromises driven by the needs of the specific application.

The advantage and attraction of products being able to deploy NNs in local devices puts pressure on SoC companies to offer devices with an NPU acceleration option. However, this is where the difficulties begin for these companies. The problem is that a SoC design cycle from concept to release is typically 2-3 years, whereas the state-of-the-art in NN techniques are changing every few months. The risk that the chip designer runs is that they can very easily lock-in a processing architecture that does not adequately foresee the evolution of the state-of-the-art. So by the time the chip is released (and keeping in mind that the chip will also need to service the market for many years after that), the NPU design can be almost useless at accelerating the NNs developers wish to deploy. Here, the same issues we described in previous sections with regards to memory bandwidth and ISP limitations similarly apply; the processing pipeline either encounters an efficiency bottleneck or simply cannot execute the network needed by the developer. Neither is a good outcome.

There are two approaches to NN acceleration that are evident in today’s market: (i) companies that design and build their own custom NPUs or (ii) those that re-badge and repurpose a different type of processing accelerator IP to be more compatible with the kinds of processing NNs need. The latter is achieved largely through refactoring of toolchains, allowing them to ingest NN models. For instance, DSP’s, GPUs, or Vector Processors are often wrapped in an “AI” marketing envelope and positioned as full-blown NPUs.

Looking behind the curtain, we must remind ourselves that accelerators (in general) are one of the primary ways that SoC companies differentiate their products. Therefore it is certainly understandable as to why semiconductor companies employ these “wrapping” tactics. However, it can be very difficult for end-customers and algorithm developers to peer through the marketing haze. It’s an area full of traps, consisting of misleading statistics and things left unsaid until it’s discovered by a developer far too late, at which point the hapless customer finds themselves in the classic “walled garden” where the financial outlay, training and code commitment already made into developing with a chosen SoC means changing devices is too expensive and risky.

When a developer like Seeing Machines, who needs to stay abreast of the state-of-the-art and develop new kinds of networks, requires an operator or network connectivity structure that the NPU (or whatever accelerator) was not designed to support, the result is either that the developer cannot take the best-in-class algorithm implementation to production or a

compromise in performance must be made due to processing inefficiencies that need to be accounted for somewhere else in the pipeline.

In order to illustrate this point, Table 4 below shows a number of SoCs performing the same accelerated function for our custom Gaze Tracking task (which looks at one or both eyes on a driver or front-seat passenger and determines where the person is most likely to be staring). Of note, the Gaze Tracking task incorporates a NN with an unorthodox architecture. Also included in the table, for reference, is the same task executed on an A53 CPU (code fully optimised) along with the Occula NPU running on the FOVIO Chip.

	HW Accelerator	Advertised "TOPS"	Latency (ms)
SoC5	DSP	3.00 TOPS	7.2 ms
SoC6	DSP	2.00 TOPS	57.3 ms
SoC7	Custom Vision Accelerator	1.00 TOPS	9.9 ms
SoC8	Custom Vision Accelerator	0.05 TOPS	32.6 ms
FOVIO Chip	Occula NPU @250MHz	0.03 TOPS	3.1 ms
CPU Only	1x A53 + NEON @1.1GHz	0.003 TOPS	9.5 ms

Table 4: Accelerated implementations of Gaze Tracker in various SoCs

The execution latencies show how the different designs of the accelerator hardware yields an enormous impact on the processing efficiency outcome. Note that in this table we show latency in milliseconds as **the most critical measure**, not Tera-Operations Per Second (TOPS). We intentionally show the variation in TOPS numbers for the different NN Accelerators to highlight the danger of assuming that it's possible to use this number alone for any meaningful and practical resource planning. You may recall the adage that throughput (or TOPS) can be bought, but latency must be earned!

For example, a 60Hz processing rate requires all the computing tasks to be finished within 16.6ms; similarly a 30Hz processing rate requires completion within 33.2ms. To put the Gaze Tracker task into a wider perspective, the plot in Figure 9 below indicates the overall time allocation for Gaze Tracking in our pipeline with the 60Hz and 30Hz processing deadlines overlaid as dashed lines.

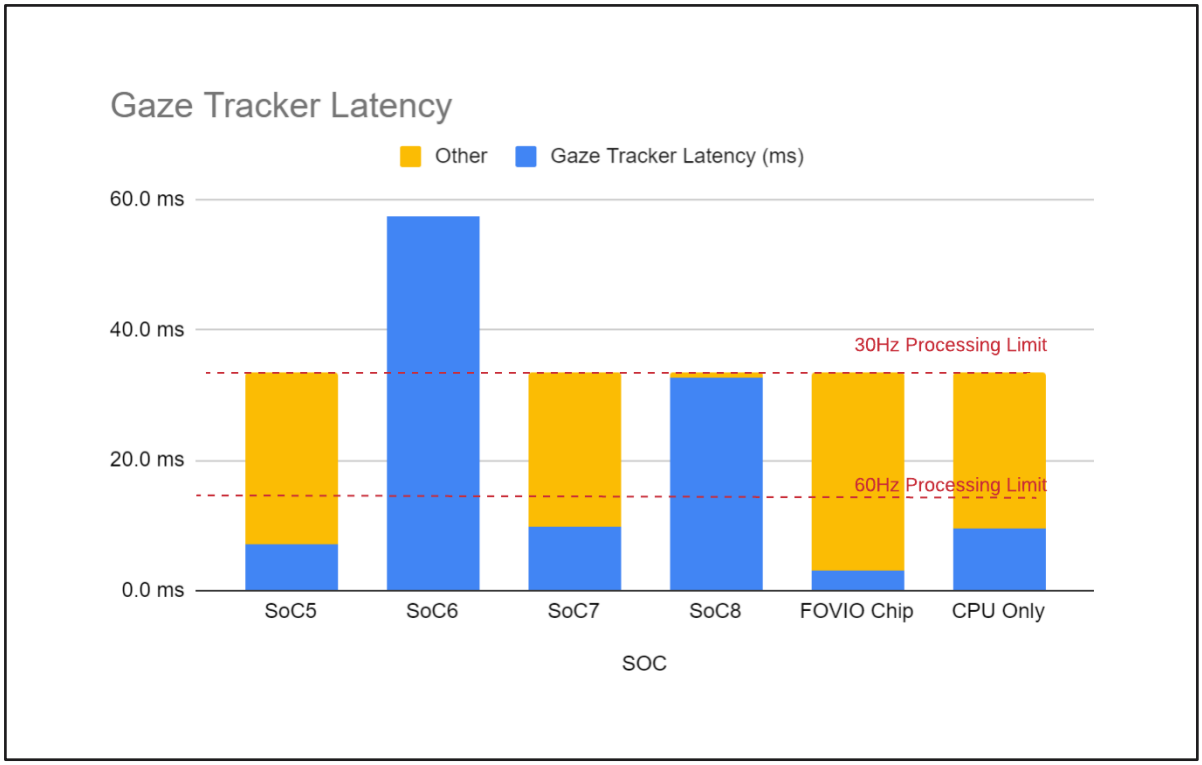


Figure 9: Example Gaze Tracker versus Unallocated Compute Bandwidth

In this figure, each yellow bar indicates the remaining *time-slots* available for scheduling all other tasks in the pipeline after the Gaze Tracker. This illustrates that the goal of completing the task as quickly (and efficiently) as possible is what always matters. It's all about ensuring that the yellow bar is as large as possible, maximising the chance that there are sufficient resources remaining to execute the rest of the DMS feature stack, which at time of device selection, may also contain customer-specific features that are yet to be developed.

The comparison shows that there are some accelerators and chip architectures that are very poorly matched to this particular part of our pipeline with two of them clearly unusable and a third no better than the CPU. Note also that the A53+NEON core is reasonably efficient at running this particular processing task, largely because the Gaze Tracker workload has been thoroughly optimized at Seeing Machines for ARM processors (thanks in large part to the NEON SIMD extensions available in ARM instruction sets).

In any standalone DMS embedded system design, the processing budget allocated to the overall driver monitoring engine may be as little as one A53 CPU core, requiring the full algorithm stack to always execute in under 33ms (considering not the average but the worst-case execution pathways). Therefore allocating 9.5ms to this task would likely not be a great use of the available bandwidth (9.5ms/33.3ms = 29% of the CPU utilised). Hence, the need for Accelerators is clear - to offload processing from the CPU wherever possible.

The data in **Figure 9**, along with the data presented earlier on the memory bandwidth for RGB-IR processing, shows the relative performance of different SoCs for a specific task with the aim of highlighting that chip marketing does not reflect reality. The fact is that different silicon designs deal with NN processing demands in very different ways with a large range of potential outcomes. In some scenarios, the chip designer, or the algorithm developer might get lucky

and create a good match, but in the embedded world such a fortuitous outcome is painfully rare.

In summary, embedded performance is largely about the match, or degree of fit, between software and hardware. For embedded NPUs, it is how well the NPU design is matched to the specific NN technology that is of critical importance.

The Co-Design Approach

At this point, it is natural to ask if a NN can be reworked to better match a specific accelerator design? To some degree the answer is yes, NNs can be reworked by machine learning experts to better fit specific accelerator designs. However, this path is by no means easy or cheap to do, and the outcomes are hard to predict in advance.

A better approach is software-hardware **co-design**. This is a process whereby the NPU and the NNs are developed in lock-step, with two design teams working closely together to build a single optimized “system” of hardware and software. Note that this is the antithesis to the general-purpose NPU approach discussed earlier.

An issue with this approach is that the NPU design will need frequent iteration to keep up with the evolution in the NNs. Seeing Machines chose FPGA technology for it’s FOVIO chip in order to be able to rapidly explore the cost vs. performance space for DMS solutions, knowing the market time-frames involved and also with the knowledge that we would need to deeply optimize the embedding in order to be competitive for standalone DMS solutions.

FPGA technology allows for an “algorithm-first” design approach to the embedding challenge and to some extent, frees our AI developers to incorporate the most powerful neural network techniques and still have those networks run very efficiently.

Application Specific NPUs

The term “application specific” refers to silicon designs that are intended to serve limited and specific purposes, and are therefore able to avoid bloat in the design that comes from attempting to support other unknown applications. Taking the Co-Design Approach naturally leads to solutions that only do what they need to, and therefore tend to be well matched to the domain of the application they target.

In contrast, a “generic” NPU is one that will be designed to serve a wide array of (often disparate) network architectures and operators. When chip manufacturers decide the requirements for their generic NPUs, the features and capabilities they end up adopting are guided by the state of the art in the research field and the popular networks that are being used by their end customers at that point in time. There are no “standards” per se when it comes to NNs that would make their requirements capture a straightforward process, but there are certainly a number of long-running threads in terms of research and innovation which target certain domains of application, such as resource constrained embedded systems. Two

recent examples are MobileNetV3 from Google[3], and RegNet from Meta[4] (formerly Facebook).

When defining a generic NPU architecture, the challenge is to build a small and efficient silicon design that can anticipate the future of NNs despite the high uncertainty in what is still a rapidly evolving landscape of research and development in both industry and academia. Given this fairly early stage of the innovation cycle in NNs, chip vendors are forced to cast a wide net to ensure that in 2-3 years' time, when their first silicon samples are available, the application domain has not seismically shifted. This of course leads to considerably larger capacities and capabilities that are invariably suboptimal and inefficient.

On the other hand, application specific NPUs can be defined in a very narrow and targeted manner. The key decisions around what to include and what to exclude are driven by the algorithm design. This generally means culling the set of mathematical operators to be as well matched to the proprietary algorithms as possible. There is still a need for some headroom and flexibility in order to incorporate future changes, but that is a perfectly feasible tradeoff when taking the Co-Design Approach - which is what we have been doing for many years at Seeing Machines. A simple example is that the operational parameters of an application specific NPU can be made runtime programmable without growing silicon area.

[3] https://arxiv.org/pdf/1905.02244.pdf
[4] https://arxiv.org/pdf/2003.13678.pdf

Occula NPU

The Motivation - Understanding Humans

Today, products that require human interaction are designed to almost always use buttons for input, whether software or physical. Buttons are an extremely simple and effective interface solution in many situations, however anywhere there is a non-trivial interaction required between a machine and human there also exists the potential for buttons to become the *barrier* to using the machine. Behold the TV remote control.

Many products have the potential to be far simpler to use if they could somehow just detect what the human wants or needs without the human needing to find and press the right button. This is simple enough in theory, but in practice, detecting what a random human might want or need, requires an AI that (to some degree) can interpret far more naturalistic human commands.

Today we see some signs of success in voice-agent technologies, which are sophisticated enough to interpret human speech usefully, allowing some products to be controlled using voice alone. Voice is primarily good for commands, however many people feel a sense of discomfort issuing voice commands to a machine, and this is especially true where multiple people are present and who may already be talking. Voice commands are powerful, but are not always appropriate, and certainly not a silver bullet. Instead they simply create another interface option. Machine commands can also be issued by making physical gestures, using the hands, face and eyes. These can be used on their own, or combined with voice, to further enrich the interface sophistication.

However, while many devices can be improved from having voice or gesture commands, the greater value comes from the machine having access to richer *contextual information* about the individual and thereby dynamically adapting the interface to the real-time context.

We believe that context is truly the key to more intelligent machine interfaces. Context can be assembled from the following four sources (i) the machine's present "state" (e.g. driver is attempting to enter a travel destination), (ii) the environment (e.g. a highway, road-scene, night etc), (iii) the person's own digital information (e.g. their past destinations, friend's addresses etc), and (iv) *the person themselves*. If the machine can tell if a person is frustrated, angry, upset, calm, overwhelmed, confused, relaxed, sleepy, sleeping, jovial, drunk, engaged in a task, (the list goes on and on), then it will be able to far better serve the user while needing fewer commands.

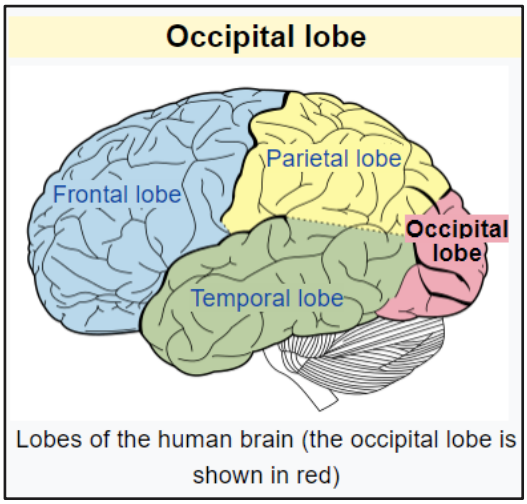
Today's world is only just beginning to witness the value that real-time human context can provide to next-generation HMIs. The first systems are now appearing in luxury cars, with the Mercedes Benz S-Class being the primary example. Here vision, audio and haptic sensory interfaces (used for both input and output) are placed around the driver, placing them "in the loop" on many interface pathways. The result is a natural feeling, highly intelligent interface that enormously simplifies access to an otherwise extremely broad set of signals coming from the vehicle's systems and the road environment.

In summary, Occula has been developed to address the limited application scope of not only DMS, but more generally of "understanding humans". While that might sound like a very broad

application, it still leads to a common set of specialized NN algorithms and therefore optimized methods of execution and so a distinct embedded advantage can be found over more general purpose NPU designs. Despite being designed and built from the ground up for DMS solutions, the Occula NPU design when married with Seeing Machines DMS algorithm stack, may offer performance advantages to a far wider range of products - any product that is (i) price or power sensitive, and (ii) can yield an advantage from understanding contextual information about humans. We leave it to the reader to imagine the possibilities.

SM-DETECT and SM-TRACK

Understanding humans begins with detection and measurement of human bodies. Humans have evolved to be highly social animals and the biology of the human brain tells us that one of the most critical components of the human body, to "visually understand" or comprehend, is the face.



Lobes of the human brain (the occipital lobe is shown in red)

This is evidenced through study of the brain region known as the occipital lobe[5], located at the back of the skull which directly receives nerve impulses from the optic nerve. The occipital lobe contains a sub-region called the occipital face area[6]. Experiments by neuroscientists show that this appears to be a neural network cluster entirely dedicated to low-level detection of facial features.

Inside the face itself, arguably the most important feature to detect is the eyes. This is because eyes reveal the all-important information as to where a person is looking. This cue, combined with the context of the scene, provides extremely valuable insight as to what another person might be thinking about at any given moment and is therefore a crucial component for advanced social interaction. In the brain, the detection of eye features is believed to be performed in a dedicated region known as the superior temporal sulcus[7], whereas the higher-order conversion of face and eye temporal-spatial information into emotional cues occurs in the amygdala and the prefrontal cortex, which also have countless other roles.

To an embedded engineer, it appears that human brains have simply evolved dedicated NN "hardware" acceleration for detecting and tracking facial features. The reason for this evolutionary step is perhaps that facial comprehension is a processing intensive yet essential task when in a social context and the brain, as an organ (or computer), already consumes a lot of calories. Evolution appears to have selected not only for bigger brains, but for brains that are extremely efficient at performing the things they need to do everyday.

Inspired partly by witnessing these specialized biological networks, Seeing Machines has developed similar optimized processing pathways for detecting and tracking human body

[5] https://en.wikipedia.org/wiki/Occipital_lobe
[6] https://en.wikipedia.org/wiki/Occipital_face_area
[7] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4346170/

parts. These functional units are the “base of a pyramid” of the DMS processing hierarchy, and together form the “perception” layer needed for human-machine social engagement.

SM-DETECT is a “fast path” for detecting faces, facial features, torsos, arms, hands etc, and also can be trained to detect other kinds of objects which may appear near human bodies, such as sunglasses or phones. The algorithm methods chosen for detection are by no means the highest in terms of detection accuracy, but rather are a compromise between speed and accuracy, carefully selected to be maximally compatible with hardware acceleration and to minimize processing power.

The FOVIO chip firmware executes the SM-DETECT pathway periodically to scan the scene for body-parts, and these detections are used to support the frame-by-frame tracking of the vehicle occupants.

SM-TRACK is a similar idea. A fast-path, but for locating and tracking body-parts over groups of frames. This pathway takes advantage of the knowledge that body parts can only move so far between video frames, and performs localized detection of body-parts based on a prediction of where they are likely to be in the latest image, incorporating a model of the basic human form to do so. SM-TRACK saves the majority of processing bandwidth versus the standard solution of using a NN to fit a 3D (or “4D”) model to every frame of video. Once more, the algorithms used are derived from over fifteen years of internal evolution, leading to a carefully chosen compromise between speed and accuracy in the environment of a vehicle cabin.

Seeing Machines makes no claim that these algorithms are the best performing in terms of their ability to detect and track humans in an image, but we believe that they are the best compromise between “good enough” tracking performance and that all important metric, the Cost of Processing.

Classifying Human State

In the human brain, complex higher order reasoning tasks are performed by the prefrontal and frontal cortex, which appear to be at the pinnacle of the network hierarchy and where our human consciousness largely resides.

To an embedded engineer, the frontal cortex region of the brain looks somewhat like a general purpose NPU. This is perhaps where a large variety of different networks help us navigate the complex puzzle of survival in a social world, with each network consuming as input the state from the higher-bandwidth perception layer beneath.

Similarly, Occula is designed to supports more general NNs, especially of the type and size that take results from the SM-DETECT and SM-TRACK perception layer and perform higher-order classifications; for example, to detect a microsleep in the last two seconds of eyelid data, or to infer the level of drowsiness over several minutes of observation of the whole driver.

To do this, Seeing Machines engineers surveyed the set of classifier algorithms required for a modern DMS, looking at operators, the model sizes, the precision of the numbers and the required execution time budget, and designed Occula to “best-fit” all the known combinations, within a silicon resource budget.

Occula Efficiency Outcomes

This section examines the design efficiency of Occula against three types of tasks that commonly occur in DMS systems: (i) face detection, (ii) face tracking, and (iii) inference of MobileNet (a commonly used NN architecture).

Benchmarking Approach

As explained, Occula is not a generic NPU. rather it is acceleration hardware that is matched (or co-designed) with NN-based algorithms which, in-turn are designed to run efficiently on it. **So it is important to think of Occula as both an NPU and a set of algorithms, which are indivisible.**

Therefore for an efficiency test to have meaning, it is important that we compare the Occula NPU+NN **system** and compare it to other potential **NPU+NN systems**, such as those that might be developed by competitors to Seeing Machines.

To do this we created face-detection and face-tracking benchmark algorithms on the candidate NPUs that are well suited for those NPUs and which we feel would represent the development path taken by any expert machine-learning engineer tasked with implementing a state-of-the-art DMS software solution on those devices.

The idea is to compare what Occula achieves “out of the box” as an application-specific design vs what might be the outcome if tasked to develop face-detection and face-tracking NNs on any other general purpose NPU. We argue that this approach closely represents the reality of porting “hardware agnostic” DMS software to a random SoC, which is what our competitors must do.

Face Detection Task

For face detection, our benchmark employed a MobileNet SSD-V2 using TensorFlow, trained on a repository of face images. Single-Shot Detectors (SSDs) are, at the time of writing, the state-of-the-art for general 2D object detection using NNs. Additionally, the MobileNet NN architecture produces small model sizes that are commonly used in high performance embedded solutions. We felt this approach to be the most likely method chosen by a machine learning engineer developing a face detector for a DMS solution. Both SM-DETECT and the MobileNet SSD-V2 networks were run against 320x320 resolution images.

Face Tracking Task

For face tracking, our benchmark employed an open-source implementation of a 3D “face alignment” technique, originally presented at ECCV 2020. The paper is titled “Towards Fast, Accurate and Stable 3D Dense Face Alignment” and pursues maximally efficient fast real-time face-alignment using NN techniques. Both the ECCV paper[8] and the 3DFFA_V2 GitHub project[9] are co-authored by Jianzhu Guo[10] a highly cited and respected expert in the field of face processing. In our benchmark, we measure the time taken for the face alignment stage, which is directly comparable to the work performed by SM-TRACK when applied to tracking a

[8] https://guojianzhu.com/assets/pdfs/3162.pdf
[9] https://github.com/cleardusk/3DDFA_V2
[10] https://scholar.google.com/citations?user=W8_JzNcAAAAJ&hl=en&oi=ao

face. Each method was executed at the native resolution of the core algorithm models to avoid image-prescaling affecting the results. The 3DFFA_V2 code was run on 128x128 resolution images, whereas SM-DETECT processed 1280x968 resolution images. Note, this gives 3DFFA_V2 less input pixels to process (and therefore a bandwidth advantage).

General Purpose NN Inference Task

For general purpose NN inferencing, the same model was executed on all target devices, including Occula. The network architecture was MobileNet-V3 with a 10MB model consisting of 8-bit quantized coefficients. The task was to perform a simple classification using ImageNet[11] data.

Target Devices

For the benchmarking we intentionally chose NPUs designs that are (i) general purpose NPUs, (ii) considered "state of the art" for their low-power embedded performance, but which are (iii) **not** found in automotive products.

The reason to exclude any NPUs that appear in automotive SoCs is because Seeing Machines also develops and supplies it's software in automotive NPUs and the performance of those devices vs Occula is commercially sensitive information. In short, we do not want to trouble any of our SoC partners.

For our studies, we chose the Google Coral Edge TPU[12] and the NVIDIA Xavier NX[13] because both Google and NVIDIA arguably lead the market on high-performance general-purpose NPU designs and have more recently released embedded versions of their chips. Both chips also have industry-leading tool-chains.

Power Measurements

Google Coral

The Coral TPU is a pure NPU device (and not an SoC with CPUs and other power consuming sub-systems). The test system provided by Google operates with a Raspberry Pi v4 with a USB Coral TPU dongle. Measurement of the power consumed by the Coral TPU is simply a matter of observing the difference in the Raspberry Pi system power when the Coral TPU is performing the processing, versus when the Coral TPU dongle is removed.

NVIDIA Xavier NX

We ran the benchmark on two different sub-components of the NVIDIA Xavier NX system, (i) the GPU (384-core NVIDIA Volta™ GPU with 48 Tensor Cores), and (ii) the Deep Learning Accelerator (NVDLA Engine) with power measurements obtained on that system..

[11] https://www.image-net.org/
[12] https://coral.ai/products/accelerator-module
[13] https://www.nvidia.com/en-au/autonomous-machines/embedded-systems/jetson-xavier-nx/

Benchmarking Results

	Device				
Performance Aspect	Google Coral Edge TPU	NVIDIA Xavier NX GPU	NVIDIA Xavier NX DLA	FPGA Based FOVIO Chip with Occula R8.2	ASIC or ASSP with Occula R8.2
Transistor node (nm)	12	12	12	28	12
NPU Estimated Die Area (mm^2)	2.5	70	22	6.5	0.9
NPU Peak Arithmetic Capability (TOPS)	4.0	0.8	11.0	0.013	0.026
NPU Peak Power (Watts)	2.0	15	9	0.8	0.3
NPU Arithmetic Efficiency (TOPS/Watt)	2.0	0.05	0.76	0.02	0.09
Face Detection Task Efficiency (FPS/Watt.mm^2)	22	0.3	3.1	27	634
Face Tracking Task Efficiency (FPS/Watt.mm^2)	132	1.2	4.2	36	644
MobileNet Task Efficiency (FPS/Watt.mm^2)	63	0.4	1.4	3.4	80

Table 5: **Benchmark outcomes**

Table 5 above compares design efficiency by considering (i) how fast the NPU can execute the task in frames-per-second (FPS), (ii) how much additional electrical energy is consumed by the NPU to perform the calculation (Watts) and (iii) the silicon area consumed by the NPU core in square millimeters (mm^2). Note that for Occula, the benchmark was performed in the context of the FOVIO Chip which is a 28nm FPGA device, whereas the Google and NVIDIA devices are considered 12nm parts and therefore should be far more power efficient.

The column on the far right is a modelled projection of the performance outcomes if the Occula design is deployed into a 12nm part, which gives a better “apples-to-apples” efficiency comparison. The details of this model are discussed in the following section.

An interesting outcome is that the Google TPU significantly outperforms the NVIDIA device for all three tasks. We note that the Google device is clearly optimized for execution of quantized NNs, while the NVIDIA device retains full floating point support.

The results reveal the well understood industry phenomenon that application-specific designs are usually able to outperform general-purpose designs (when compared in the application-specific field) by at least an order of magnitude. Indeed, this is why co-processors exist. Perhaps the deeper realization here is that while NPUs themselves are specializations away from more general-purpose CPUs, most NPUs (even embedded ones) remain highly generalized designs which can be out-performed by the application-specific co-design approach.

Modelling Occula NPU Performance at 12nm

There are three components to modelling Occula performance in various semiconductor technology nodes for comparisons.

First, the processing rate measured in frames per second (FPS) of any given algorithm running on Occula is largely dependent on the clock frequency and the deterministic pipeline of Occula. While there is some variance due to memory bandwidth policies at the system level, this is covered by including a small safety margin.

Second, the silicon area required by Occula depends on the number of gates and the total RAM bits instantiated for internal buffers and caches. These are fully determined from synthesis results in the ASIC design flow.

Third, the power draw of Occula depends on the leakage properties of the technology node, silicon area, clock frequency and the dynamic switching activity of various sub-modules inside Occula. These factors and their subtle interplay are succinctly explained by Stillmaker and Baas[14]. As they point out in their paper, if we have the values for clock frequency, area and power at a particular node (say 22nm), then there are scaling equations we can use for estimating the equivalent quantities at a different node (e.g. 12nm).

We have listed our modelling results in the right-hand column of Table 5 above (shaded in gray). We conclude that at the 12nm node, Occula will require 0.9 mm^2 area maximum, and 0.3 W peak power at 125 °C junction temperature for worst-case switching activity (which is in fact extremely difficult to hit given the near certainty of a memory bottleneck arising at the system level well before this point). The performance of Occula at this node is expected to yield at least a 500 MHz clock frequency after taking into account silicon aging.

It’s actually possible to increase that frequency considerably due to the way Occula is smartly pipelined for ease of timing-closure, but it has to be kept in balance with the speed of DDR memory interface because that is the precious resource in a chip (as we’ve elaborated earlier). If Occula is clocked excessively higher than 500 MHz then it’s likely to be waiting for data from

[14] https://dx.doi.org/10.1016/j.vlsi.2017.02.002

the DDR memory and not actually *using the silicon real estate efficiently*. We discuss this topic further in the next section.

Value of Silicon Real Estate

When developing a new SoC, the most important initial consideration is the silicon area, as it directly correlates to the marginal unit-cost and therefore the sales price of the device. As a reference point, an SoC processor targeting the standalone DMS market (dual to quad A53 class processors) may have a silicon area in the range 15-20 mm^2 at the 16nm node. This is a realistic estimate that we can use to illustrate the importance of being efficient with the silicon design and how it is utilised - which we think of as the *value of silicon real estate*.

An example silicon area breakdown of a complete SoC product at 16nm is shown below in the pie chart of **Figure 10**. This simple illustration is intended to put into perspective the relative differences in the value of silicon utilization of the various functions within a device. In the figure below: the Vision Accelerator may be a custom programmable coprocessor, a DSP or an NPU; the Video Codec is a H.264 codec and the ISP is capable of RGB-IR processing along with other viewable video stream processing; the CPU subsystem has quad 64-bit cores (each implementing vector extensions) with 512 KB of shared L2 cache.

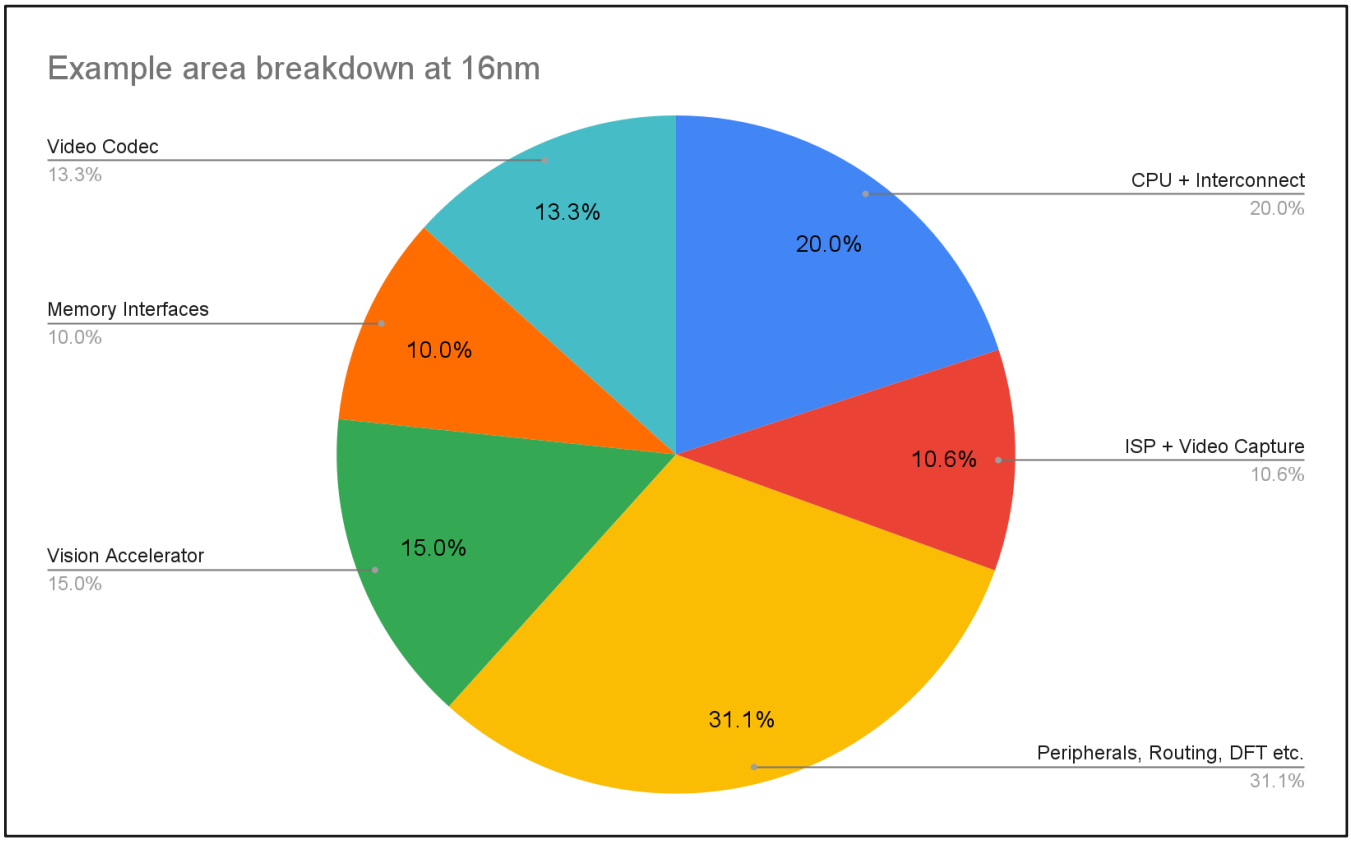


Figure 10: Example silicon area breakdown at 16nm technology node

It should be noted that there are actually three different “accelerated” subsystems in this example device, each with the purpose of relieving the CPU tile of computationally heavy workloads. These are the H.264 codec, the ISP and the Vision Accelerator. So why have these

items been chosen to be hardened into silicon? The answer is simple but not always obvious. It comes down to the following key considerations:

1. Is the task going to be used by most applications that use the chip (i.e. it is not dead silicon for a large portion of customers)?

2. Is there a clear silicon reduction advantage in hardening the function (i.e. it is a lower silicon cost than running on the CPU)?

The main downside when hardening any function is that you lose flexibility. After all, software in a CPU is the most flexible solution possible. Generally speaking, functions that are "standards" based and ubiquitous are perfect candidates for hardening as there is little chance that the SoC manufacturer misses something in their definition - an H.264 encoder for example is an ideal candidate.

Given the above breakdown of the silicon area, an insightful analysis can now be done of the silicon cost for frequent and **real-time tasks**. For instance, how *efficiently* is the silicon utilized when compressing a Full HD 1080p video stream in a hardened H.264 Encoder versus doing it purely in software on a CPU?

We ran a simple experiment for the data shown below in Table 6. A hardened H.264 Encoder clocked at 200 MHz can keep up with a 60fps incoming video stream even though only half of the frames are RGB. In other words, the encoder must still be better than 16.6ms frame latency despite processing at 30fps because the remaining frames are IR and they will arrive at that same rate; otherwise the encoder becomes massively inefficient by forcing every RGB frame to be buffered in DDR memory first (as we discussed earlier in the paper).

On the other hand, one A53 core clocked at 800 MHz takes 130ms to encode one frame with purely memory-to-memory dataflow and no blocking storage - hence it fails to meet the 30fps requirement altogether! We used the open-source x264 utility that has been heavily optimized for NEON SIMD extensions in the ARMv8 instruction set. This data demonstrates that hardening the standard H.264 algorithm can provide the same feature at better than 5x the efficiency with respect to an A53 core when it comes to weighing up the value of silicon real estate.

	Area (mm^2)	Latency (ms)	Relative Efficiency
H.264 Enc. @200MHz	2.4	15.6	5.2
1x A53 (*) @800MHz	1.5	130.0	1.0

(*) estimated size of one A53 core and cache scaled from publicly available information (https://en.wikichip.org/wiki/arm_holdings/microarchitectures/cortex-a53)

Table 6: Relative Silicon Efficiency of H.264 encoding in hardware versus pure software. Area estimate is for 16nm technology node

Now let's focus on the Gaze Tracking task (which executes our proprietary SM_TRACK network) for the same analysis related to Occula NPU. **Table 7** compares an A53 core (with NEON SIMD extensions) executing the SM_TRACK network to an Occula NPU running the same workload. We're comparing the one-shot latency for each and then scaling that latency ratio to the silicon area of an A53 core to derive a relative efficiency (same as Table 6). What

this shows again is how efficient a hardened Occula is at executing our Gaze Tracking algorithm with respect to a fully optimized software version running on an A53 core.

	Area (mm^2)	Latency (ms)	Relative Efficiency
Occula @ 500MHz	1.0	1.6	12.3
1x A53 (*) @800MHz	1.5	13.1	1.0

(*) estimated size of one A53 core and cache scaled from publicly available information (https://en.wikichip.org/wiki/arm_holdings/microarchitectures/cortex-a53)

Table 7: Relative Silicon Efficiency of Gaze Tracker (SM_TRACK) for Occula with respect to an A53 core. Area estimate is for 16nm technology node

The data above makes it clear that Occula running at 500 MHz (the typical clock for an ASIC implementation) is making at least 12x better use of the silicon real estate than an A53 core running at 800 MHz when it comes to a real-time task like Gaze Tracking. This ratio is to be expected and the relative performance, along with previous data illustrated in this paper, show how versatile an A53 core with NEON SIMD extensions actually is as an embedded processor. It is worth noting that both the A53 core and the Occula NPU can certainly perform multiple other tasks as per their designed architectures. The point still remains that critical real-time workloads which burden the CPU should be accelerated directly in silicon for maximum efficiency.

Concluding Remarks

Building a DMS for the automotive industry is as much an art as it is a science. There are multiple strategies that can be deployed, and considerable uncertainty associated with each of them. For many years, Seeing Machines has been at the forefront of designing DMS solutions that satisfy the most demanding requirements and market pressures for low cost, low power, high performance and high volume applications.

Our strategy for navigating this landscape of art and science has been centred on solving the embedding challenge by deploying the co-design approach. Now in its eighth generation, we have crafted the Occula NPU to optimise DMS technology, based on careful analyses and insights gained from our own algorithm inventions and collective expertise in this field.

Looking ahead to the future possibilities in this segment of the automotive market, and also fanning out to other industries that adopt DMS technology as a matter of necessity - we see opportunities emerging for pushing the cutting edge of this technology much further. We have a world class team to exploit those opportunities, thanks to our deep knowledge and skills (spanning the full stack from top to bottom) when it comes to designing DMS products.