https://doi.org/10.2352/EI.2023.35.16.AVM-113

OpTIFlow – An optimized end-to-end dataflow for accelerating deep learning workloads on heterogeneous SoCs

Shyam Jagannathan, Vijay Pothukuchi, Villarreal Jesse, Kumar Desappan, Manu Mathew, Rahul Ravikumar, Aniket Limaye, Mihir Mody, Pramod Swami, Piyali Goswami, Embedded Processors Business, Texas Instruments
Carlos Rodriguez, Emmanuel Madrigal, Marco Herrera

Abstract

A typical edge compute SoC capable of handling deep learning workloads at low power is heterogeneous by design. It hosts multiple compute modules such as real-time IPs for capture and display, hardware accelerators for imaging, computer vision, deep learning, multimedia codecs, GPU for 2D/3D visualization and CPU cores such as ARM and DSP for general compute. Every participating compute module transacts using common resources such as system interconnect, L3/L4/DDR memory systems to seamlessly exchange data between them. A careful orchestration of this dataflow is important to keep every producer-consumer at full utilization without causing any drop in real-time performance. The software stack for such complex workflows can be quite intimidating for customers to bring-up and more often act as an entry barrier for many to even evaluate the device. In this paper we propose techniques developed on TI's latest TDA4V-Mid SoC, targeted for automotive and industrial applications; designed around ease-of-use and device entitlement class of performance out-of-box using community popular open standards such as GStreamer, OpenVx, OpenCV, TFLite-RT, ONNX-RT, Neo-AI-DLR runtimes.

Introduction

Deep-learning based solution is today's defacto approach to solve various real-world problems using embedded devices. As show in Figure 1 realizing simple tasks such as object detection, image classification, semantic segmentation requires a careful consideration and design while using low power heterogenous SoCs to ensure a balanced cost-power-performance tradeoff. Software for such demanding use-cases can be highly complex to bring-up and maintain, it can take many man-months to realize and prove quite an entry barrier for customers to quickly evaluate and prototype. Typically, a middleware is the bridge between high level software stack and low-level device firmware. It is important to choose a popular open-source middleware which provides a familiar programming interface to developers and also aims to provide device entitlement class of performance out-of-box. In this paper we discuss the merits of community popular GStreamer [1] and OpenVx [2] middleware which helps bridge this gap and help customers quickly put together performance optimized demos in minutes. TI's latest TDA4V-Mid [3] based Edge AI starter kit comprising of ARM A72, R5F, TI's C71x, C66x DSP, accelerators for ISP, lens-distortion-correction, multi-scaler, noise-filter [4], dense-optical-flow, stereo-disparity-engine, multimedia accelerators for H.264/H.256 codecs and GPU, is the ideal solution for both automotive and industrial analytics market offering 8 TOPS of deep-learning performance and capable of handling 4-8 multiple cameras of varying resolution all under 10 watts of power.

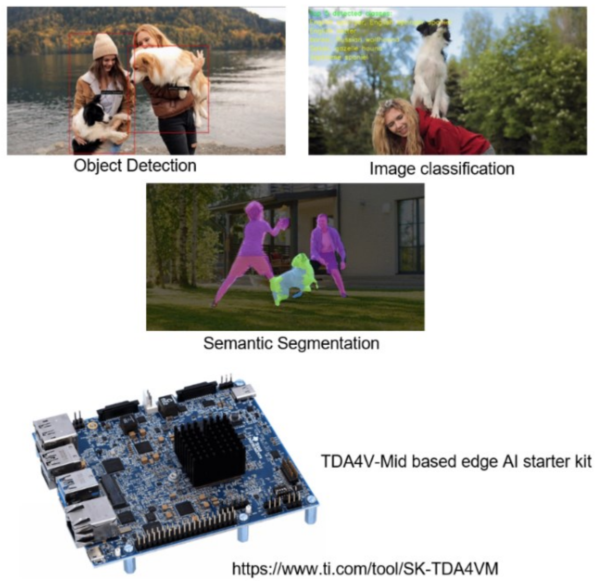
Object Detection
Image classification
Semantic Segmentation
TDA4V-Mid based edge AI starter kit
https://www.ti.com/tool/SK-TDA4VM

Figure 1. Example illustration of various deep learning tasks possible on TI's latest TDA4V-Mid based EdgeAI starter kit

Typical camera-inference-display dataflow

A typical "camera-inference-display" pipeline will involve capturing raw images from a sensor via real-time CSI interface, followed by ISP processing to extract RGB or NV12 images, followed by an optional lens-distortion correction for fish-eye lenses to provide a rectified image. This could branch to either human vision or machine vision pipeline which require image resizing and pre-processing before submitting the frame for deep-learning based inference. The output of inference is visualized with some post-processing and this either sent to real-time HDMI/eDP display or encoded for storage or streaming. Every single operation described will be mapped to the most optimal and available accelerator which basically splits the operations over multiple cores. A designated host core constructs this signal-chain to manage the real-time data and orchestrate multiple accelerators to get the desired output. The key aspect here is to build an optimal dataflow without copying image buffers between multiple compute cores. Each compute core might be operating at different frequencies and it becomes challenging to keep each producer-consumer pair adequately balanced to meet use-case performance goals.

OpenVx is a standard designed by the Khronos group to help orchestrate varying computer-vision functions as an acyclic graph. Figure 2 illustrates a typical OpenVx application graph which realize such inference use-case. The host core would construct a single graph comprising of multiple "nodes", where each node represents an imaging or vision operation. A node is an instantiation of a "kernel" which could either run on the same host core or a remote target. The target kernel would in-turn call a low-level driver to interface with an accelerator. TI's OpenVx implementation optimizes the signaling and data-buffer exchange between these remote targets. While it helps abstract and simplify multiple aspects of multicore processing, it still requires the application developer to understand some coarse grain details of what the framework offers to build an optimized graph. It particularly gets challenging when graphs have to be intersected at multiple intermediate points, mandating a solid understanding of the middleware and its minute nuances. This at times becomes an entry barrier for customers, the cost associated here being the time taken to comprehend the complexities of the middleware and also intricacies of the hardware to realize such simple usecases.

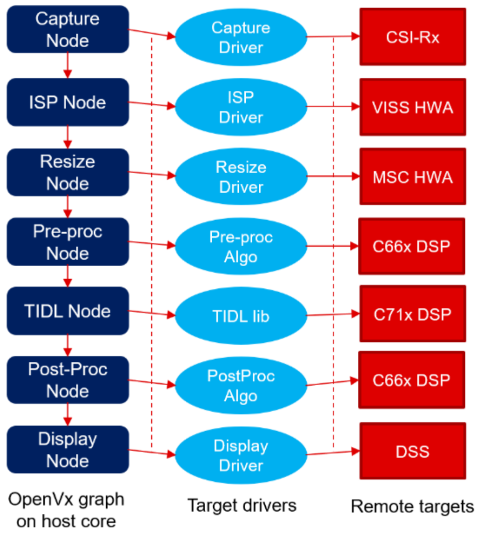

Figure 2. Typical OpenVx based capture-inference-display pipeline illustrating single acyclic graph of interconnected nodes which remote core/accelerator processing

Interplay of open source frameworks and SoC

It is quite important for silicon vendors to abstract out both hardware and software complexities to help customers quickly evaluate their device. Making the programming model more familiar to a group of engineers who are exposed to a rich eco-system of open source frameworks such as GStreamer, TFLite-RT [5], ONNX-RT [6], OpenCV [7] is the key to a successful design win. While many such open-source software stacks are offered off-the-shelf to run on ARM cores, it's the responsibility of the silicon vendors to provide hooks to their underlying accelerators and compute cores in a seamless way to differentiate performance.

As shown in Figure 2, customers would love to build use-cases with various type of inputs such as USB camera, smart camera with on-board ISP, RAW cameras which requires on-chip ISP, compressed video/image files from disk, compressed bitstreams streamed from a network etc. These varying inputs can be pre-processed and fed to a choice of popular deep-learning runtime such as TensorFlow-RT, ONNX-RT or TVM based Neo-AI-DLR [7]. The output of inference would be post-processed to apply some visualization such as drawing boxes on detected objects using popular image processing/computer vision libraries such as OpenCV. This is followed by either displaying the results on the screen or encoding the same for storage or streaming. While providing this flexibility improves ease-of-use, this shouldn't come at the cost of performance. This establishes as an interesting problem statement to offer open-source software stack with device entitlement class of performance.

Flexible use cases

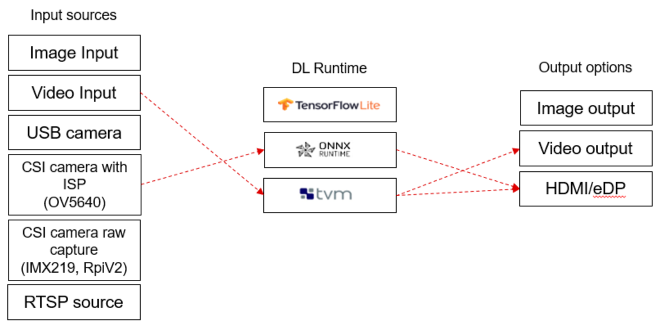

Figure 3. Flexible runtime configurations to select between multiple inputs, inference runtime and outputs

Flexible analytics dataflows using GStreamer

GStreamer is an industry popular streaming media framework which allows users to define flexible data pipelines mainly to perform audio and video processing. A typical GStreamer application comprises of multiple participating "elements" or "plugins", each responsible for executing a certain function. They are typically connected back to back with data flowing from "source" to "sink". The framework automatically negotiates the capabilities of each participating element, allocates buffer pools and starts the pipeline for streaming. Each element typically runs in its own thread and maintains data and control queues to maintain the order of processing but still function asynchronously. The GStreamer framework also provides a way to define "custom elements" by deriving its base classes. While GStreamer offers complete flexibility and ease-of-use, it can get extremely complex to design a "zero-buffer-copy" pipeline, especially when the participating elements are running on different cores/accelerators. TI's OpTIFlow solution combines the best of both GStreamer and OpenVx to provide an optimal dataflow for camera based analytics enabling open-source stack to run efficiently on heterogeneous SoCs like TDA4V-Mid.

Figure 4 illustrates OpTIFlow in action connecting the industry popular RaspberryPi v2 [8] raw camera with Sony's IMX219 sensor to TDA4V-Mid's ISP followed by a multi-scaler operation to feed the human vision and machine vision pipelines. The color-convert and pre-processing operation is offloaded to C66x DSP before calling a choice of open-source deep-learning runtime for inference on C71x DSP. The inference output is then post-processed using OpenCV on ARM and presented to the display in a windowed fashion using the multi-scaler hardware accelerator.

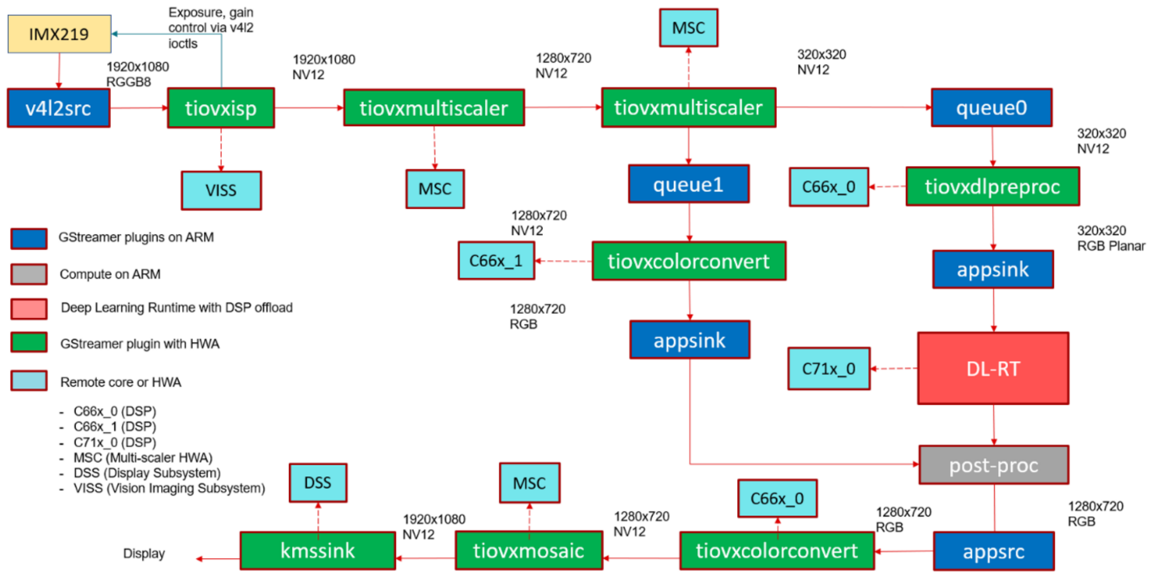

Figure 4. Typical GStreamer based capture-inference-display pipeline illustrating multiple compute cores and accelerator processing

OpenVx based custom GStreamer elements

As shown in Figure5, a typical OpenVx node comprises of a host module and a target kernel. The host module is an interface between the application developer and compute module. The target kernel calls the low-level drivers to work with the hardware to get the desired functionality. The OpenVx frame work takes care of managing input and output buffers, handling IPC between host core and target core, maintaining desired cache coherency and ensuring data integrity [9]. A call to the host module is non-blocking, the host can go to IDLE state or execute other tasks while the target kernel gets busy executing some function with the hardware.

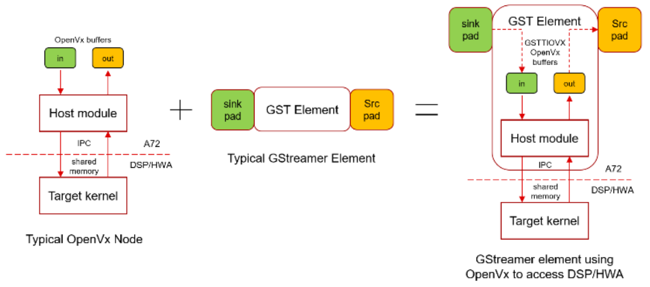

Figure 5. Interplay of OpenVx and GStreamer elements for flexibility and efficient processing

Whereas a typical GStreamer element comprises of one or multiple sink and source pads attached to the body of the element. The pads are responsible for negotiating capabilities of the peer elements and manage data buffer pools between the elements. The body of the element performs a desired function. This function can either be executed on the same host core or be offloaded to a remote core. The solution proposed in this paper, combines the best of both OpenVx nodes and GStreamer elements. While GStreamer elements provides the flexibility of running them as independent threads, the OpenVx node is used to offload efficiently to the remote target. Figure 5 also show how a GStreamer element can call an OpenVx node. Each element creates one-graph-one-node pipeline as opposed to one-graph-multiple-nodes in the OpenVx case. The buffers exchanged between sink and source pads are passed as-is to OpenVx input and output buffers respectively. The OpenVx framework takes care of passing the buffers to the remote targets by applying the appropriate cache operations.

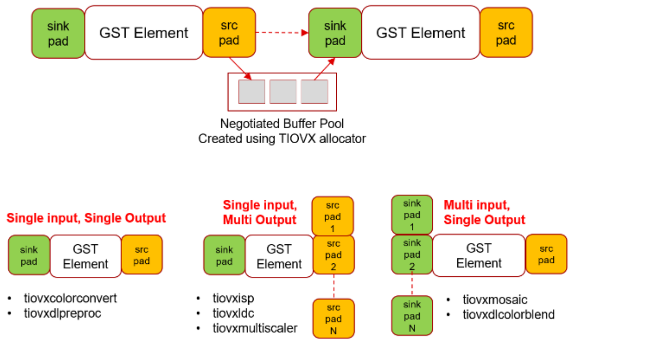

Figure 6. Extending GStreamer base classes to create custom base classes to handle zero-buffer-copy between elements. Template classes to handle different input, output scenarios

As the top-level interface is GStreamer we need to extend the GStBufferPool base class to create custom classes which help allocate buffers in the common pool exchanged between host and target cores, and also help glue the buffer data-structures between GStreamer and OpenVx nodes. As shown in Figure 6 we created 3 derived classes GstImageBufferPool, GstTensorBufferPool and GstRawImageBufferPool, to help transact with vx_image, vx_tensor and tivx_raw_image OpenVx data structures. These extended classes use the OpenVx based memory allocator to create the buffer pool negotiated between custom peer elements. This is how zero-buffer-copy is achieved. To handle different combinations of input and outputs we also extend the GstBaseTransform class to create different extended classes to handle single-input-single-output, single-input-multiple-output, multiple-input-single-output usecases.

Batch processing with custom GStreamer elements

While OpenVx based graphs have multiple nodes in a single graph, the custom GStreamer elements create multiple graphs with a single node. This configuration provides flexibility but it increases the number of interrupts issued to host core especially in the muti-channel scenarios. Each participating element issues a completion interrupt for every frame processed. If a pipeline comprises of M elements and we process N channels then host core will receive MxN elements. OpenVx graphs can handle multiple channels by creating vxObjectArray [10] which enables batch processing. Standard GStreamer elements does not provide an option to perform batch processing, so we created new GstTiovxMux and GstTiovxDeMux elements as illustrated in Figure 7 which helps batch multiple inputs and take advantage of OpenVx ObjectArrays under the hood. Peer elements which understand batch processing can be linked together to construct an end-to-end chain. This way we potentially reduce the number interrups from MxN to Mx1. If there are any standard elements in-between then it should be guarded between a pair of demux and mux elements as defined by the usecase.

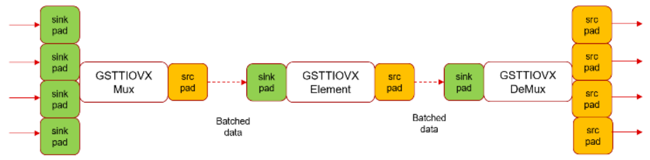

Figure 7. Interplay of OpenVx and GStreamer elements for flexibility and efficient processing

DLInferer Gstreamer element to support multiple DL runtime

OpTIFlow also supports working with popular deep-learning runtime framework such as TensorFlow-RT, ONNX-RT and TVM based Neo-AI-DLR. In construction the design is similar to custom GStreamer elements. We have defined a DLInferer class which abstracts different runtimes. Users can provide a pre-imported models to the desired runtime which gets either completely or partially offloaded to the DL accelerator and fall back to host ARM for unsupported layers [12]. The backend to these runtimes is again an OpenVx node which runs on C71x DSP target. Tensor objects are exchanged between the custom GStreamer elements and DL runtime without any buffer copies as the memory is again allocated from the same OpenVx allocator.

Results

Table I below shows total time taken to execute a GStreamer pipeline with custom elements in milliseconds(ms) followed by a breakup of inference time taken by the DLInferer element. The table also shows A72 host core loading. Mostly the processing on the host-core is attribute to post-processing using OpenCV with color-conversion and dl-pre-processing offloaded to C66x DSP. The results are discussed for 4 different DL networks executed on ONNX-RT and TFLite-RT.

Table I: Performance summary for executing different DL networks

Model name	FPS	Total Time (ms)	Inference Time (ms)	A72 (%)
ONR-CL-6360-regNetx-200mf	30.9	33.07	2.00	15.63
ONR-OD-8220-yolox-s-lite-mmdet-coco-640x640	30.5	33.13	11.02	14.39
TFL-CL-0000-mobileNetV1-mlperf	30.7	33.11	1.01	14.89
TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	30.7	33.16	5.01	11.80

Table II provides DDR bandwidth consumption in total to execute an OpTIFlow pipeline. The TDA4V-Mid SoC is paired with 4GB LP-DDR4 clocked at 4266MTS. Which at 50% efficiency at room temp provides about 8.5GB/s bandwidth. The data shown here is to execute a pipeline discussed earlier with Rpi v2 camera followed by ISP, pre-processing and one of the DL network shown. This is then followed by post-processing, windowing and sent to display

Table II: DDR throughput summary

Features	DDR Read BW (MB/s)	DDR Write BW (MB/s)	DDR Total BW (MB/s)
ONR-CL-6360-regNetx-200mf	1504	604	2108
ONR-OD-8220-yolox-s-lite-mmdet-coco-640x640	1882	756	2578
TFL-CL-0000-mobileNetV1-mlperf	1464	603	2067
TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	1553	594	2147

Table III shows further break up of tasks running on remote cores like C66x, C7x DSP and participating accelerators like MSC (multi-scaler) and ISP (for processing raw image from Rpiv2 camera)

Table III: Various target core, accelerator loading

Model name	C71x DSP (%)	C66x DSP (%)	VISS HWA (%)	MSC HWA (%)
ONR-CL-6360-regNetx-200mf	6	26	9.95	22.33
ONR-OD-8220-yolox-s-lite-mmdet-coco-640x640	32	79	9.97	21.52
TFL-CL-0000-mobileNetV1-mlperf	5	27	9.81	22.36
TFL-OD-2020-ssdLite-mobDet-DSP-coco-320x320	16	34		24.72

Conclusion

In this paper we have discussed the merits of using both OpenVx and GStreamer based approaches and how it is important to balance flexibility, ease-of-use and performance for analytics use-cases. While OpenVx dataflows optimizes execution using 1-graph-M-nodes, GStreamer based approach achieves the same using M-graph-M-nodes. The performance differences between the both approaches are negligible as maximum tasks are offloaded to a heterogeneous set of compute cores and accelerators. TI's OPTIFlow is offered currently on TDA4V-Mid class of device but the same stack will be compatible with other spectrum of analytics devices with similar hardware profiles.

References

[1] GStreamer- https://gstreamer.freedesktop.org/documentation/application-development/introduction/gstreamer.html

[2] OpenVx - https://www.khronos.org/openvx/

[3] TDA4V-Mid - https://www.ti.com/product/TDA4VM

[4] Shyam Jagannathan, Mihir Mody, Jason Jones, Pramod Swami and Deepak Poddar, "Multi-sensor fusion for Automated Driving: Selecting Model and Optimizing on Embedded Platform", AVM track, Electronic Imaging, 2018

[5] Abadi, Martín, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin et al. "Tensorflow: A system for large-scale machine learning." In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265-283. 2016.

[6] ONNX Runtime - https://onnxruntime.ai/docs/execution-providers/

[7] Neo-AI DLR - https://github.com/neo-ai/neo-ai-dlr

[8] Pagnutti, Mary & Ryan, Robert & Cazenavette, George & Gold, Maxwell & Harlan, Ryan & Leggett, Edward & Pagnutti, James. (2017). Laying the foundation to use Raspberry Pi 3 V2 camera module imagery for scientific and engineering purposes. Journal of Electronic Imaging. 26. 013014. 10.1117/1.JEI.26.1.013014.

[9] M. Abeysinghe, J. Villarreal, L. Weaver and J. Bakos, "OpenVX Graph Optimization for Visual Processor Units," 2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP), New York, NY, USA, 2019, pp. 123-130, doi: 10.1109/ASAP.2019.00-19.

[10] K. Chitnis et al., "Novel OpenVX implementation for heterogeneous multi-core systems," 2017 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia), Bengaluru, India, 2017, pp. 77-80, doi: 10.1109/ICCE-ASIA.2017.8309323.

[11] K Desappan, et.al, "CNN Inference: Dynamic and Predictive Quantization", IEEE International Conference on Consumer Electronics, (ICCE) , Berlin, 2018.

[12] M. Mathew, K. Desappan, P. K. Swami, S. Nagori, and B. M. Gopinath, "Embedded low-power deep learning with tidl," Texas Instrum., Dallas, TX, USA, Tech. Rep. SPRY314, 2018.

Author Biography

Shyam Jagannathan is an EdgeAI architect and Senior Member of Technical Staff at Embedded Processors Group, Texas Instruments. His domains of interest include DSP architecture, SoC architecture, hardware accelerators, deep learning, perception, sensor fusion localization, path planning and overall system optimization He received a master's degree in the field of Signal Processing and Communications from Illinois Institute of Technology, Chicago in 2013

Vijay Pothukuchi is an EdgeAI and Robotics Architect at Embedded Processor's Group, Texas Instruments. Hi domains of interest include, Robotics, Analytics, and middleware such as ROS, OpenVx, GStreamer and DDS.

Jesse Villarreal is a software architect for TI's heterogeneous multicore SoCs and a Senior Member of Technical Staff (SMTS) at Embedded Processors Group, Texas Instruments. He received a master's degree from the University of Texas at Dallas in Computer Engineering and has been with Texas Instruments since 2001. His areas of interest include DSP software optimization, heterogeneous multicore middleware frameworks, vision and imaging hardware accelerators, and overall system software scalability, portability, and optimization

Kumar Desappan is Senior Member of Technical Staff (SMTS) at Texas Instruments (TI) Incorporated. His domains of interest are Machine/Deep learning, image processing and computer vision algorithms with a focus on software solution for edge devices. He received Bachelor of Engineering (BE) from Anna University - Chennai in 2005

Manu Mathew is a Senior Member of Technical Staff (SMTS) at Processors Business in Texas Instruments (TI) leading the development of Algorithms and Tools for EdgeAI. His domains of interest are Deep Learning, Computer Vision, Image Processing, and Video coding. He received his Master's in Signal Processing from Indian Institute of Science (IISc) in 2000

Rahul Ravikumar is a Software Engineer working on EdgeAI SDK for TI devices at Embedded Processors Group, Texas Instruments. His domains of interests include Edge Analytics, Embedded Linux, Gstreamer. He received a master's degree from BITS Pilani in the field of Embedded Systems and been with Texas Instruments since 2021.

Aniket Limaye is a software engineer working with TI's heterogeneous multicore SoCs, at Embedded Processors Group, Texas Instruments. He received a master's degree from the Indian Institute of Technology Bombay in Electrical Engineering and has been working in Texas Instruments since 2021. His areas of interest include heterogeneous multicore middleware frameworks and system software optimization

Mihir Mody is SoC Architect lead (DMTS) responsible for roadmap and chip definition for Sitara MCU business in Texas Instrument (TI). His domains of interest are real time control, image processing, computer vision, deep learning and Video coding. He received his master's in electrical engineering from Indian Institute of Science (IISc) in 2000

Pramod Swami is Distinguished Member of Technical Staff (DMTS) at Processors Business in Texas Instruments (TI) leading the software development for EdgeAI processing. His domains of interest are Embedded systems, Digital Signal Processors, Deep Learning, Computer Vision, Image Processing, and Video coding. He received his Bachelor's degree in Electronics and communication engineering from Malaviya National Institute of Technology (MNIT) Jaipur in 2001

Piyali Goswami is a Member Group Technical Staff (MGTS) at Processors Business in Texas Instruments (TI) leading the development of Software Development Kits (SDKs) for Edge AI. Her domains of interest are frameworks, system power, performance and security. She received her Master's in Software Systems from Birla Institute of Technology and Science (BITS) Pilani in 2014.

Carlos Rodríguez is an embedded software engineer currently working as a software team lead at RidgeRun. Carlos is focused on Embedded Linux multimedia projects, including GStreamer, WebRTC, Video and Image Processing, Camera Drivers development, and Deep Learning. Carlos' work is focused on helping customers and high-tech companies to bring their multimedia products to market

Emmanuel Madrigal is a RidgeRun embedded software developer. He received a master's degree from the Costa Rica Institute of Technology in the field of Digital Signal Processing and has been with RidgeRun since 2018. His areas of interest include DSP, deep learning and embedded devices

Marco Herrera is a Computer Engineer working as an Embedded Software Developer at RidgeRun Engineering. He graduated from the Costa Rica Institute of Technology and over the past few years has been involved in multiple projects regarding multimedia processing with GStreamer as well as embedded Linux and machine learning applications.