Exploration of Resource-Efficient ML Models Targeting eFPGAs

[1]Jyothisraj Johnson, [2]Billy Boxer, [1]Tarun Prakash, [1]Carl Grace, [1]Peter Sorensen, [2]Mani Tripathi

[1] Lawrence Berkeley National Laboratory, Berkeley, CA

[2] Department of Physics and Astronomy, University of California-Davis, Davis, CA

BERKELEY LAB
Bringing Science Solutions to the World

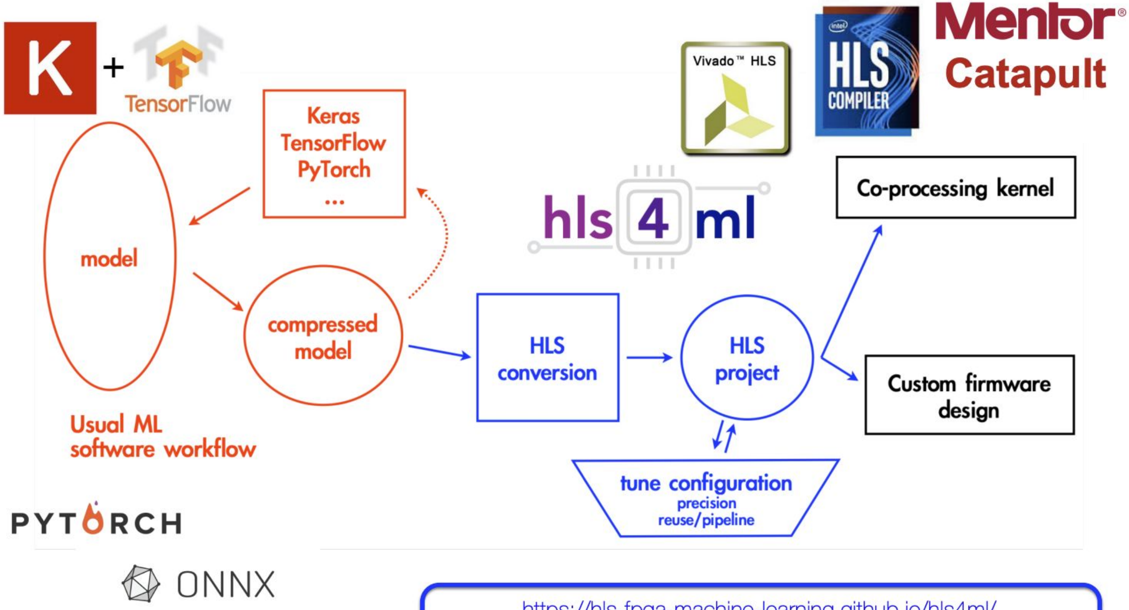
UCDAVIS
UNIVERSITY OF CALIFORNIA

The hls4ml (Conifer) Pipeline

The eFPGA pipeline mixes considerations involved for (std cell based) ASIC and commercial FPGA targets

High level synthesis (HLS) requires knowledge of target for better results

For eFPGAs, this depends on the framework capabilities → what's included?



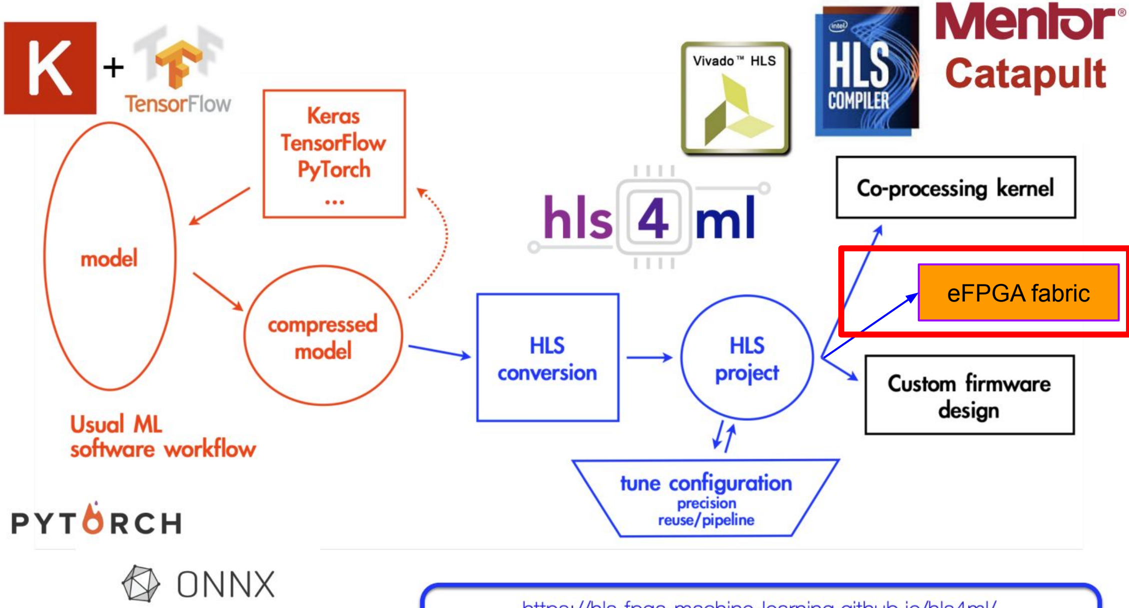
https://hls-fpga-machine-learning.github.io/hls4ml/

The hls4ml (Conifer) Pipeline

The eFPGA pipeline mixes considerations involved for (std cell based) ASIC and commercial FPGA targets

High level synthesis (HLS) requires knowledge of target for better results

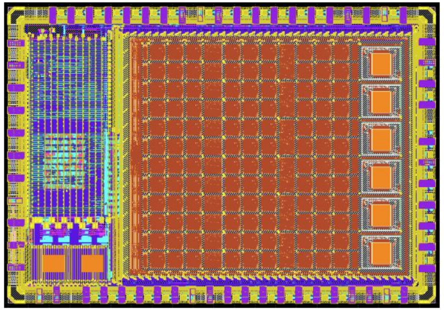
For eFPGAs, this depends on the framework capabilities → what's included?



https://hls-fpga-machine-learning.github.io/hls4ml/

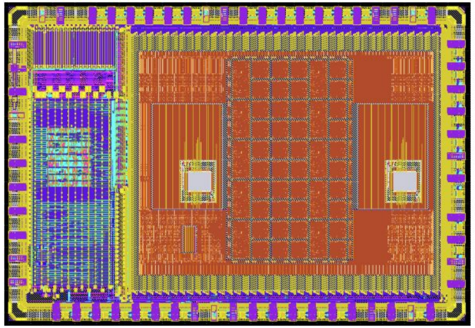
Embedded FPGAs as the Target

- eFPGAs are IP cores → integrated as part of an overall ASIC design
    - commercially available technology but expensive
    - open-source frameworks also exist now (OpenFGPA, **FABulous**)
- These cores can be reprogrammed as needed, unlike ASIC implementations of machine learning (ML) models
    - update weights/biases/thresholds of ML models **AND** make changes to the architecture
    - design core for expected range of changes to model architecture/type



(a) eFPGA_caravel_sky130
(384xLUTs, 6xDSPs, 12xRegFiles
6x1Kb BBRAMs with custom cells)

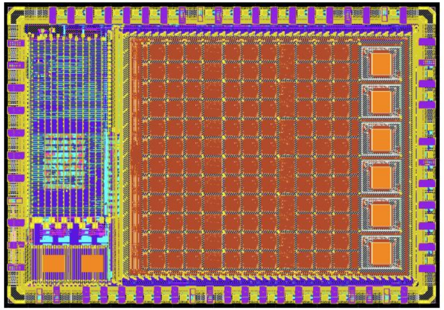
https://fabulous.readthedocs.io/en/latest/gallery/index.html



(b) eFPGA_RISCV_sky130
2xRISC-V with eFPGA
(320xLUTs, 10xDSPs)

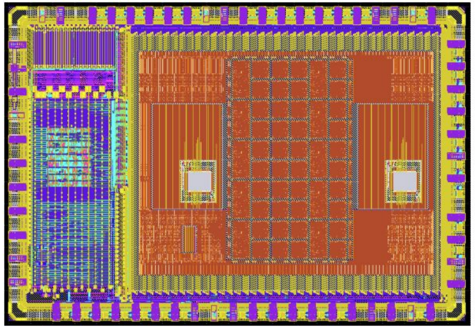
Embedded FPGAs as the Target

- Technology node, design area limitations (cost) determine possible size/complexity of ML models
    - maximum routing density, metal stack options → total block area taken up by eFPGA
        - availability and capabilities of resources → open source framework capabilities
    - maximum clk speeds achievable
- Need to focus on ML model's resource utilization as part of training
    - types and quantities of each resource



(a) eFPGA_caravel_sky130
(384xLUTs, 6xDSPs, 12xRegFiles
6x1Kb BBRAMs with custom cells)

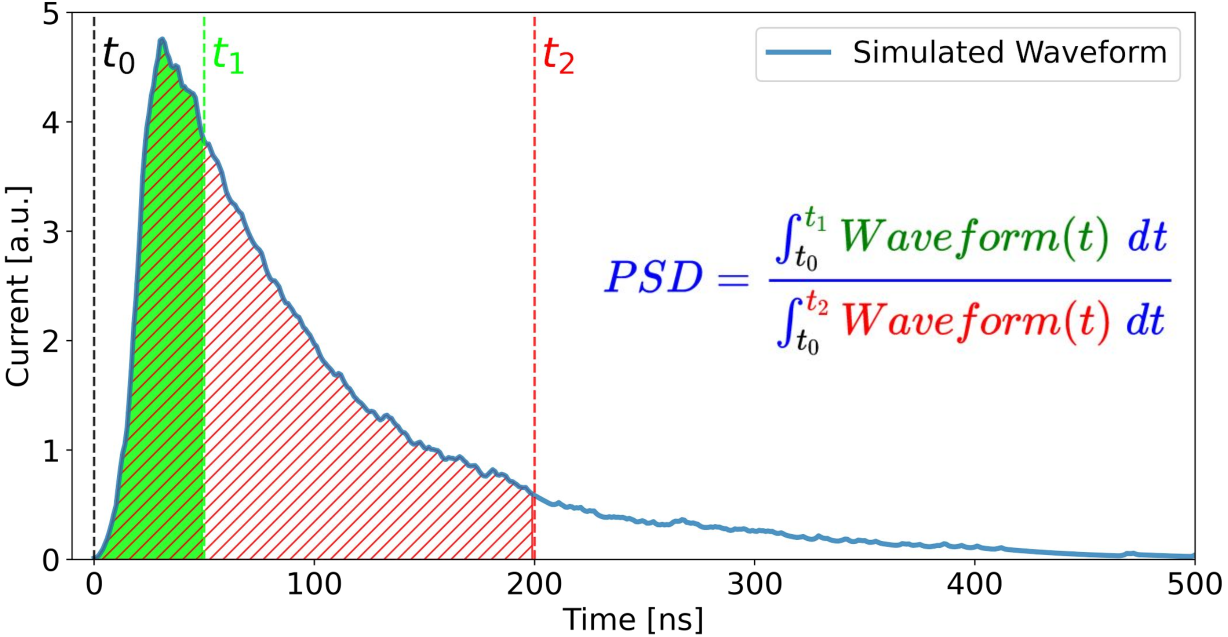
https://fabulous.readthedocs.io/en/latest/gallery/index.html



(b) eFPGA_RISCV_sky130
2xRISC-V with eFPGA
(320xLUTs, 10xDSPs)

Neutron/Gamma Classification as a Case Study

The **emission spectra** from a scintillator can be **dependant on the type of interacting particle**. Therefore, the **resultant electronic pulse** output by an optical sensor has characteristics (timing and intensity) **dependant on the particle type**. This allows for the **identification of particles using pulse shape discrimination (PSD).**



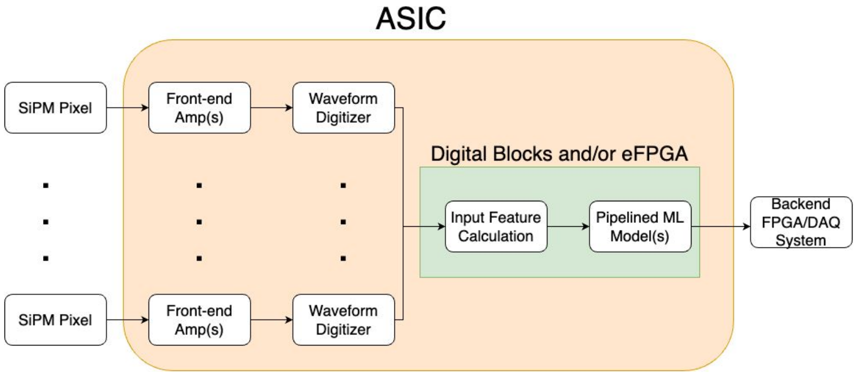
$$PSD = \frac{\int_{t_0}^{t_1} Waveform(t)\ dt}{\int_{t_0}^{t_2} Waveform(t)\ dt}$$

Why Use an eFPGA?

- Neutron/gamma classification is a requirement for applications such as (associated particle imaging; API) neutron radiography and segmented neutron scatter cameras
  - classification capability required on a channel level
  - portability and (remote) battery-operation is highly desired/required
  - eventual goal: O(100s) - O(1000s) of SiPMs that need to be individually read out for pixelated system designs at O(1MHz) event rates per channel
- For any given interaction event, not all channels are expected to receive energy depositions
  - shift classification from channel level to chip level
  - a pipelined ML model implementation is an attractive means to accomplish this
- eFPGAs are also potentially applicable to other detector systems such as hybrid scintillation/cherenkov detection schemes (i.e. THEIA)

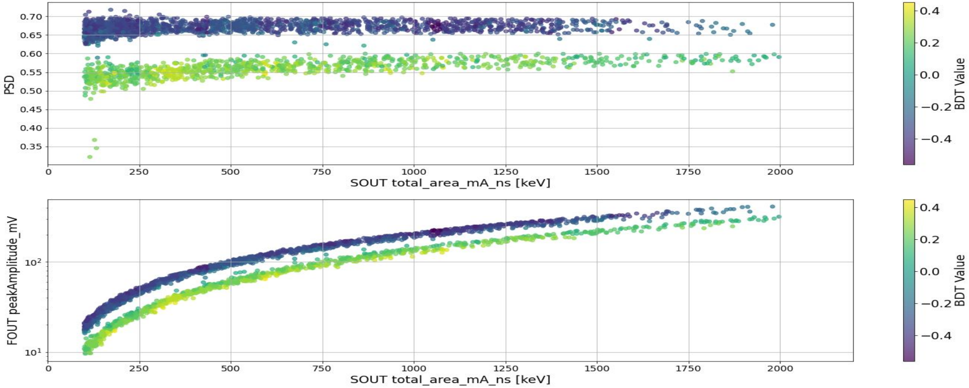
The Expected Signal Chain

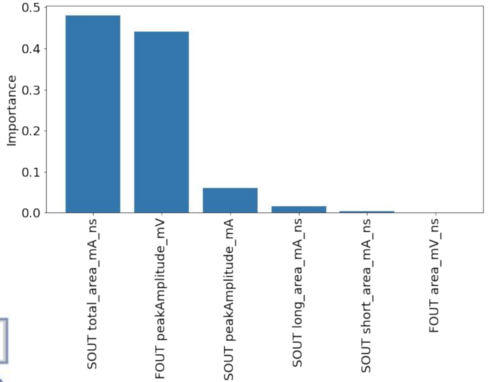
- Much like with standard cell ASIC implementations of a ML model, we can integrate more of the full signal chain on a single chip
- Include a waveform digitizer and then a DSP block to calculate the input features
  - Thus, fold in digitizer resolution and sampling rate as part of ML model specification
    - Instead of just specifying input word num_int and fractional bits as with an FPGA co-processing kernel application
  - Translation from raw ADC counts to fixed point representation w/ input feature calculation

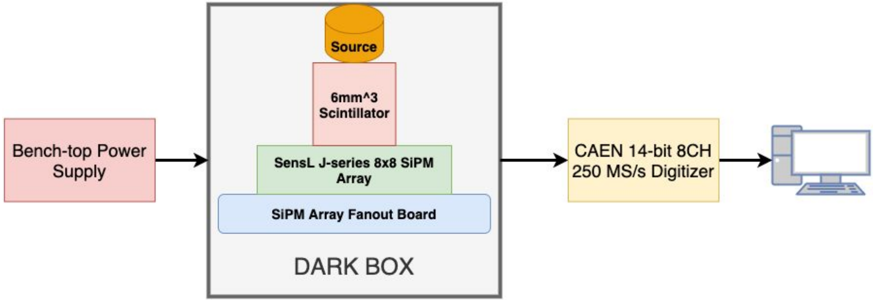


The Training Data, Test Bed, and Input Features

- Training data for study acquired with simple testbed
- Start with the basic ML architectures: BDTs and fc-NNs
  - ○ use commercial FPGA as initial stand-in during parallel development work on eFPGA ML implementation
- 5 input features selected from importance study






Bench-top Power Supply

Source

6mm^3 Scintillator

SensL J-series 8x8 SiPM Array

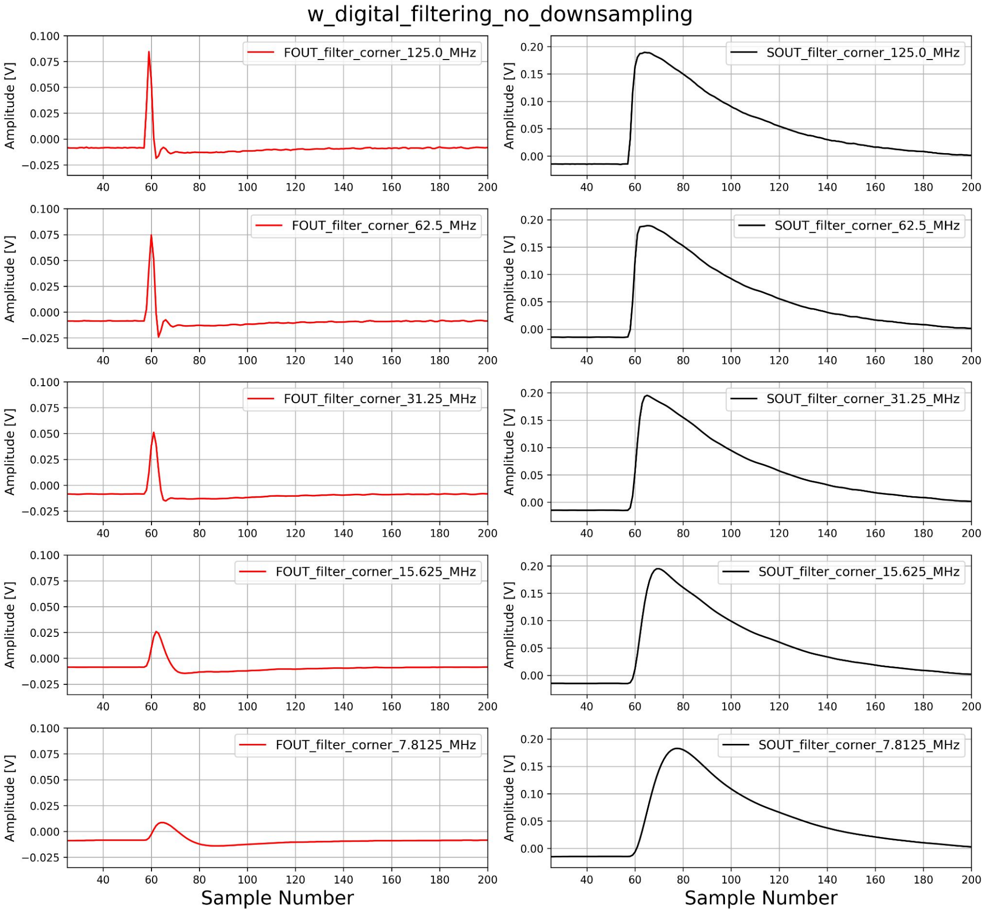
SiPM Array Fanout Board

DARK BOX

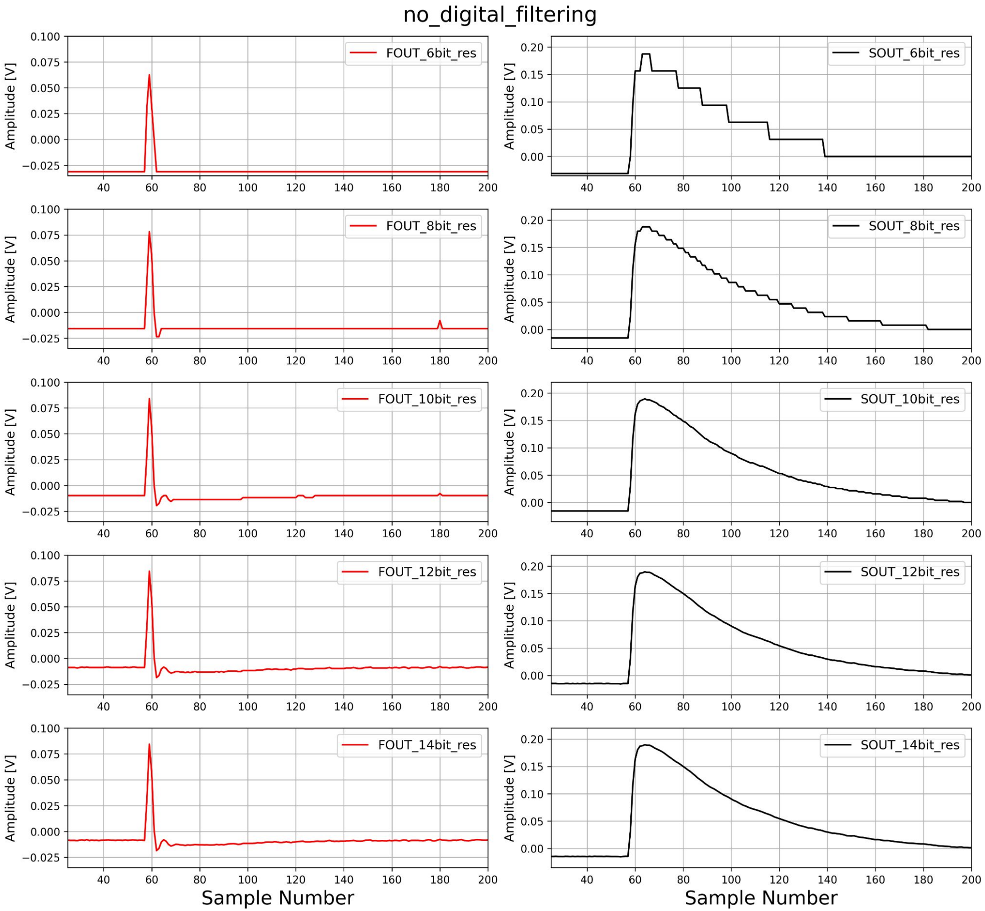
CAEN 14-bit 8CH 250 MS/s Digitizer

https://arxiv.org/abs/2209.13979

Understanding Waveform Digitizer Reqs: Sampling Frequency



Understanding Waveform Digitizer Reqs: ADC Resolution



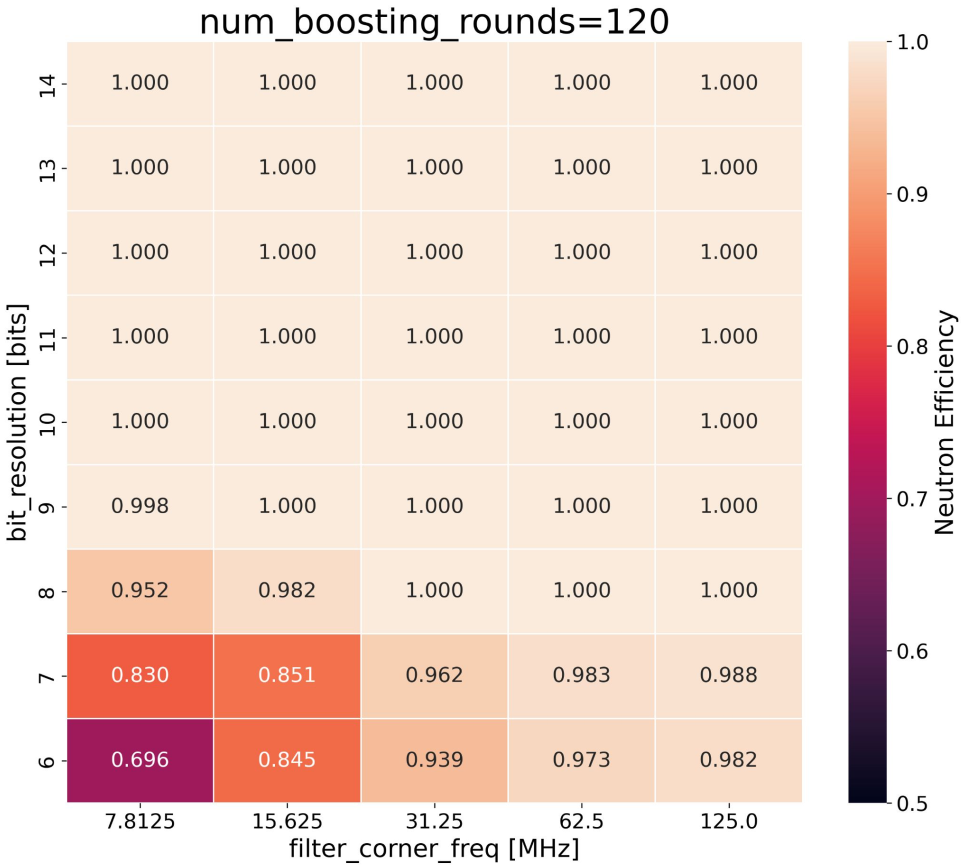
Resource-aware BDT Implementations

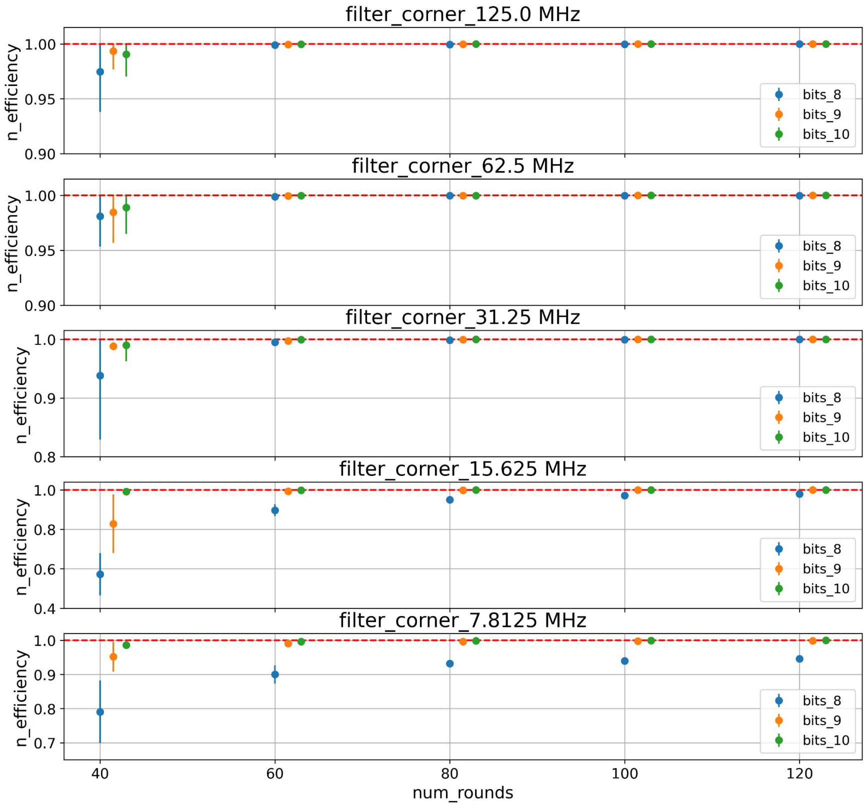
- Long list of configurable parameters but two main XGBoost BDT hyperparameters that contribute to resource usage:
  - depth of each tree (max_depth; d)
    - max_depth was set to 3
  - **number of boosting rounds (num_rounds; n_e)**
- Input feature values are specified in fixed point format (num. integer bits and num. fractional bits)
  - We set this indirectly by specifying the bit-resolution of the waveform digitizer
- Taking into account sampling rate as well → 3-dimensional parameter space
- BDTs do not require BRAM or DSP resources, only FFs and LUTs
  - Unlike fc-NNs

$$r = k_0 \cdot n_e + k_1 \cdot n_e \cdot 2^d$$

https://arxiv.org/pdf/2002.02534.pdf

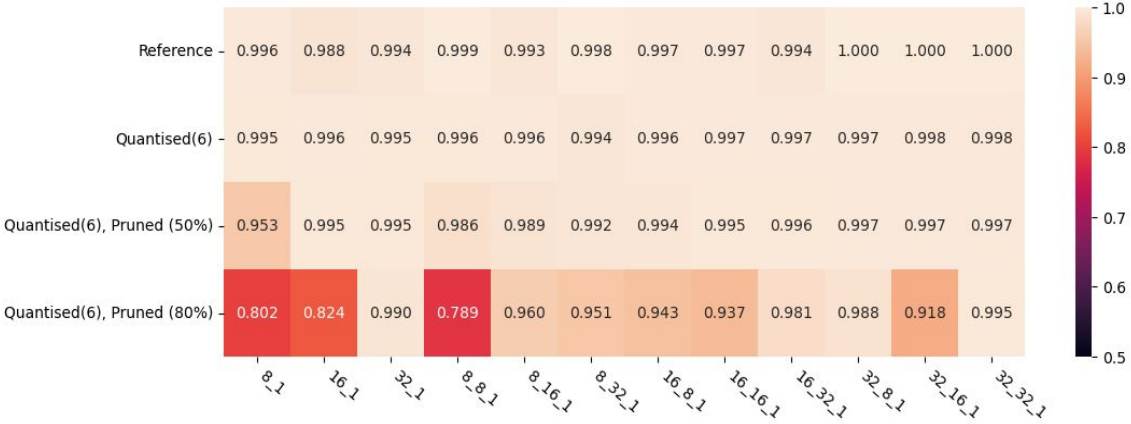
Resource-aware BDT Implementations





Resource-aware fc-NN Implementations

Hyper-parameter	Selection	Impact?	Tested?
Kernel initializer	Gaussian distribution (mu =0, sigma =1)	❌/✅	❌
Optimiser	Nadam a combination of the Nesterov Accelerated Gradient and Adam optimization algorithms	❌/✅	❌
Activation	Tanh and sigmoid produce outputs in the range (-1, 1) and (0, 1), respectively. Commonly used for binary classification.	❌/✅	❌
Loss function	Mean squared error (suitable for a regression task)	❌	❌
No. Hidden Layers	1 and 2	✅	✅
Nodes	Input fixed to 5, hidden combinations of [8, 16, 32], output to 1	✅	✅
Quantisation & pruning	Quantised to 6 bits, pruning done to 50% or 80% at frequency of 100	✅	✅
Epochs	120, selected best based on loss function	❌/✅	❌



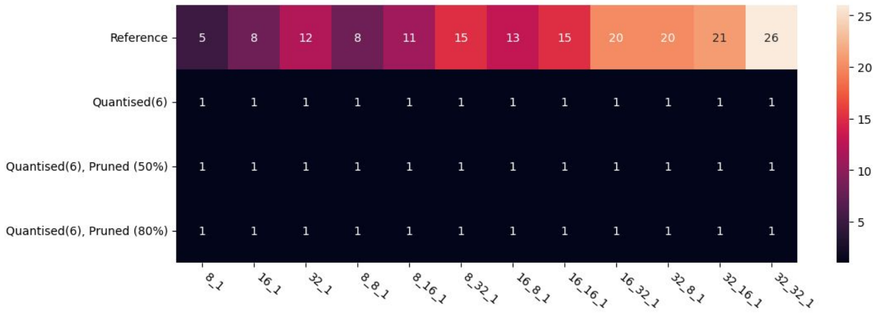
	8_1	16_1	32_1	8_8_1	8_16_1	8_32_1	16_8_1	16_16_1	16_32_1	32_8_1	32_16_1	32_32_1
Reference	0.996	0.988	0.994	0.999	0.993	0.998	0.997	0.997	0.994	1.000	1.000	1.000
Quantised(6)	0.995	0.996	0.995	0.996	0.996	0.994	0.996	0.997	0.997	0.997	0.998	0.998
Quantised(6), Pruned (50%)	0.953	0.995	0.995	0.986	0.989	0.992	0.994	0.995	0.996	0.997	0.997	0.997
Quantised(6), Pruned (80%)	0.802	0.824	0.990	0.789	0.960	0.951	0.943	0.937	0.981	0.988	0.918	0.995

There are more parameters which could contribute to the resource usage of the model

The biggest hurdle is memory resource usage for activation function!

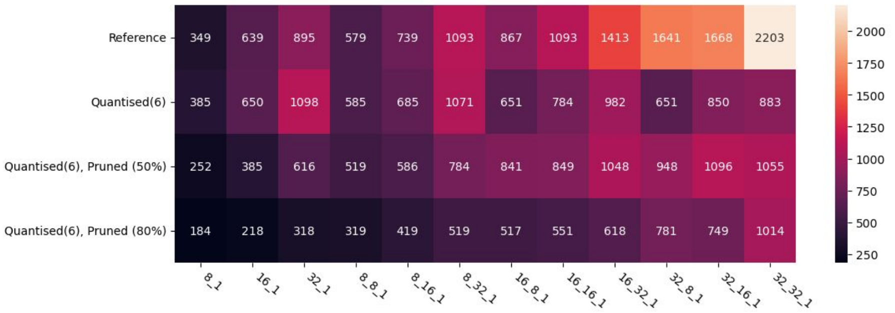
Resource-aware NN Implementations (HLS Estimates)

BRAMs



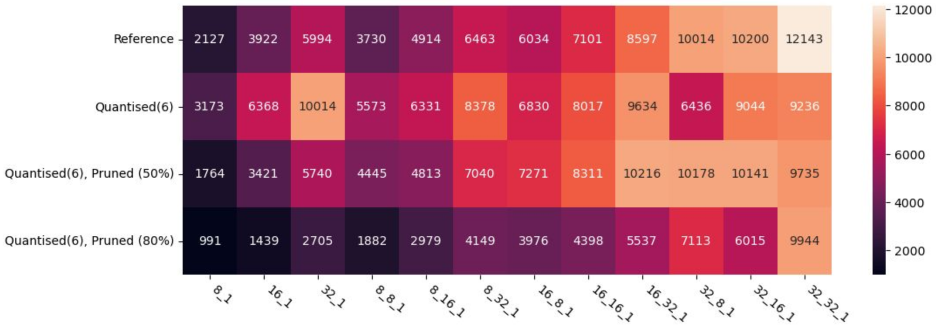
	8_1	16_1	32_1	8_8_1	8_16_1	8_32_1	16_8_1	16_16_1	16_32_1	32_8_1	32_16_1	32_32_1
Reference	5	8	12	8	11	15	13	15	20	20	21	26
Quantised(6)	1	1	1	1	1	1	1	1	1	1	1	1
Quantised(6), Pruned (50%)	1	1	1	1	1	1	1	1	1	1	1	1
Quantised(6), Pruned (80%)	1	1	1	1	1	1	1	1	1	1	1	1

FFs



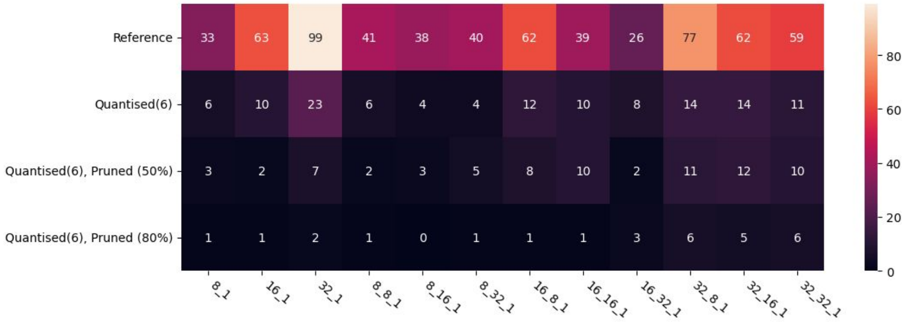
	8_1	16_1	32_1	8_8_1	8_16_1	8_32_1	16_8_1	16_16_1	16_32_1	32_8_1	32_16_1	32_32_1
Reference	349	639	895	579	739	1093	867	1093	1413	1641	1668	2203
Quantised(6)	385	650	1098	585	685	1071	651	784	982	651	850	883
Quantised(6), Pruned (50%)	252	385	616	519	586	784	841	849	1048	948	1096	1055
Quantised(6), Pruned (80%)	184	218	318	319	419	519	517	551	618	781	749	1014

LUTs



	8_1	16_1	32_1	8_8_1	8_16_1	8_32_1	16_8_1	16_16_1	16_32_1	32_8_1	32_16_1	32_32_1
Reference	2127	3922	5994	3730	4914	6463	6034	7101	8597	10014	10200	12143
Quantised(6)	3173	6368	10014	5573	6331	8378	6830	8017	9634	6436	9044	9236
Quantised(6), Pruned (50%)	1764	3421	5740	4445	4813	7040	7271	8311	10216	10178	10141	9735
Quantised(6), Pruned (80%)	991	1439	2705	1882	2979	4149	3976	4398	5537	7113	6015	9944

DSPs



	8_1	16_1	32_1	8_8_1	8_16_1	8_32_1	16_8_1	16_16_1	16_32_1	32_8_1	32_16_1	32_32_1
Reference	33	63	99	41	38	40	62	39	26	77	62	59
Quantised(6)	6	10	23	6	4	4	12	10	8	14	14	11
Quantised(6), Pruned (50%)	3	2	7	2	3	5	8	10	2	11	12	10
Quantised(6), Pruned (80%)	1	1	2	1	0	1	1	1	3	6	5	6

Picking a ML Model for First eFPGA Tapeout

- Because of the need for BRAM and DSPs in fc-NNs, we anticipate targeting BDT models for first test chip
    - final decision after merging our current parallel workflows
- Resource usage estimates are shown for commercial Artix 7-series target
    - expect number of resources to increase when targeting eFPGA fabric
    - decreased capabilities of resources in open source framework

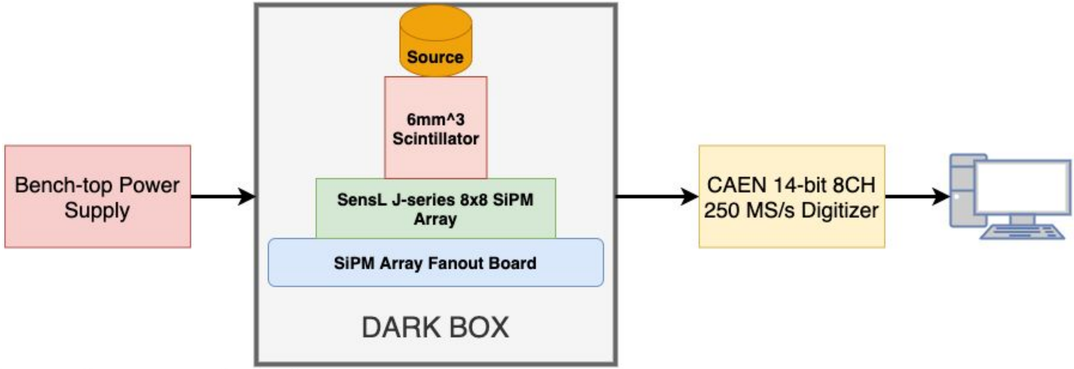
Current Status and Next Steps

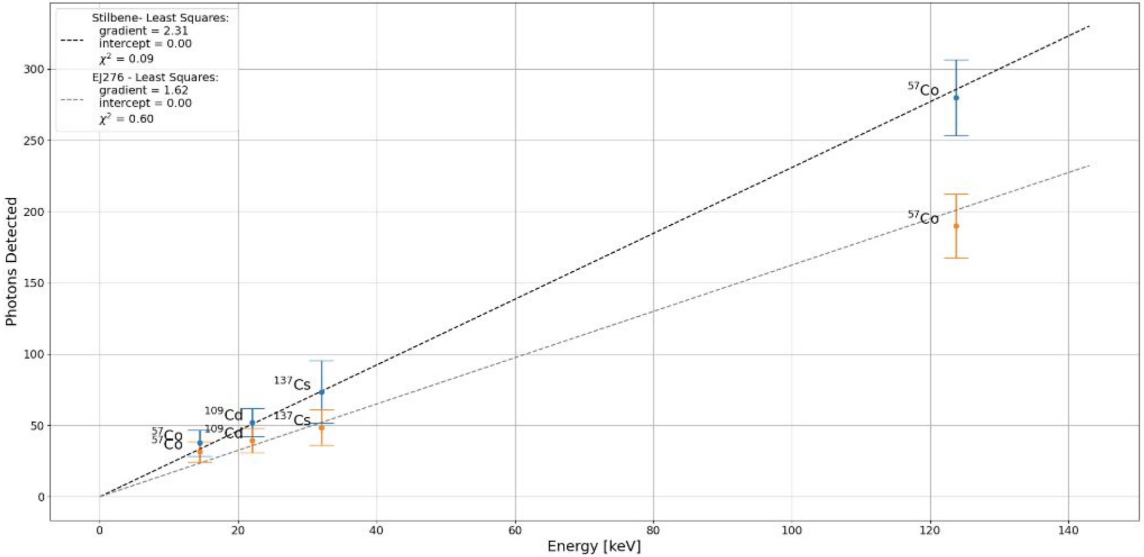
- Test chip tapeout target: early 2024 → using 130nm
  - will primarily contain the eFPGA fabric and required peripherals
- Goals with first tapeout are to:
  - first, prove functionality of the eFPGA fabric and ability to program it
  - second, build out a PCB to test/validate performance of BDT models implemented in the eFPGA targeting neutron/gamma classification using simple testbed
- Eventually, demonstrate a complete ASIC design integrating TDC, ADC, eFPGA, and required front-ends
- Also, target more complex applications as we build our capabilities
  - scale down eFPGA fabrics to 28nm
  - scale up ML model complexities (multi-class classification, other ML models)

Back-Ups

Test Bed and Energy Calibration

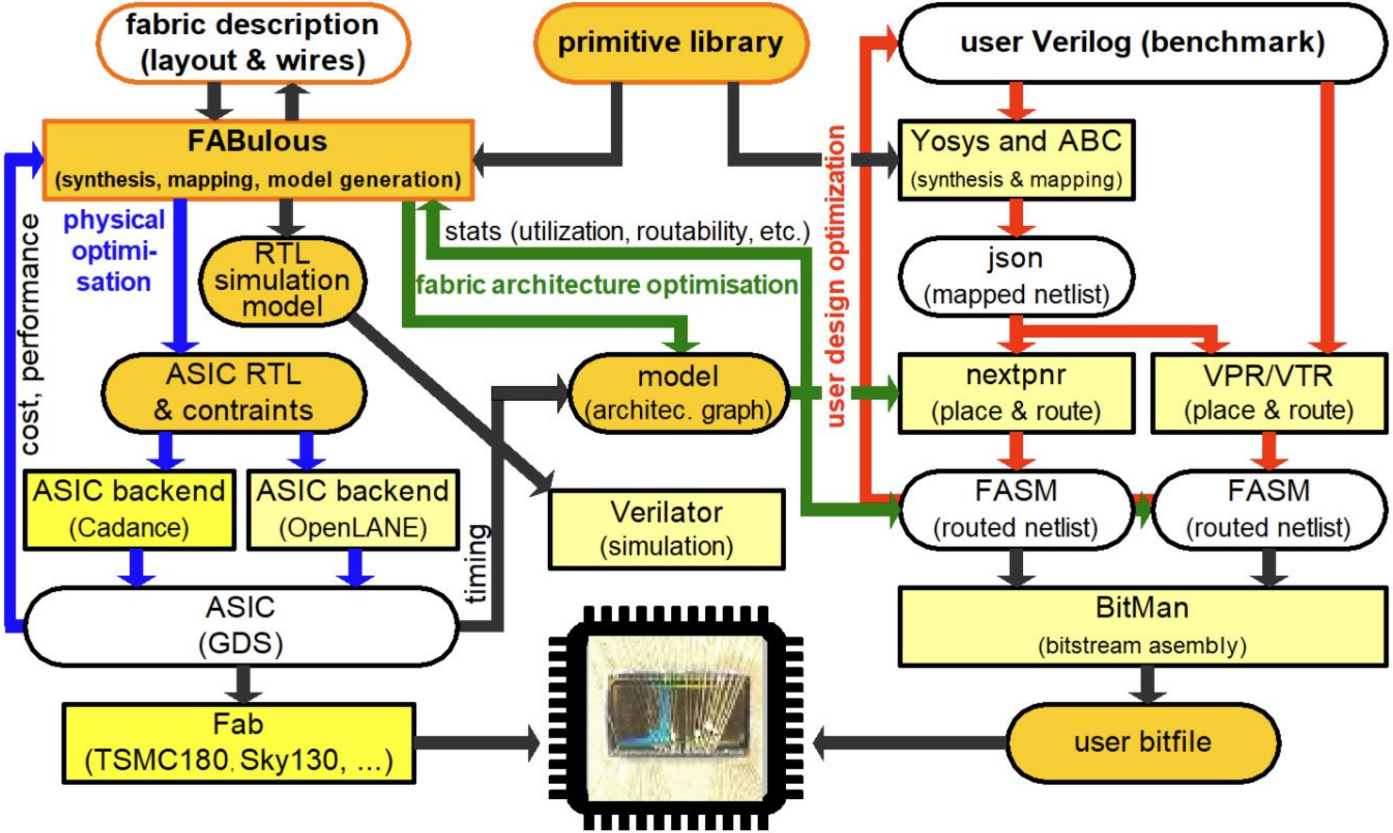
Data set 3e7 AmBe (mixed neutron and gamma source) events with 6 mm3 of scintillator coupled to a 6 mm SensL J-Series SiPM. Recording both the SOUT and FOUT traces of the SiPM.



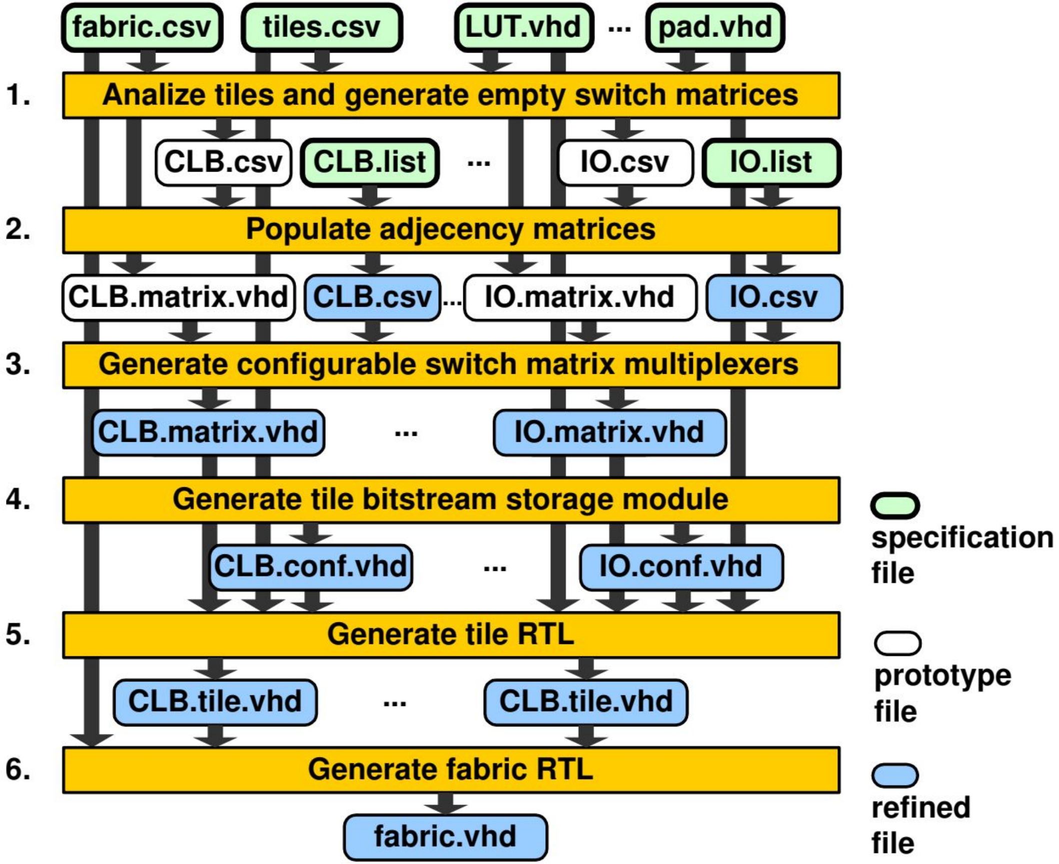


Energy calibrations performed using Co57, Cd109 and Cs137.

FABulous eFPGA framework



FABulous Flow for Generating eFPGA Fabric



eFGPA Reading Materials

- https://ieeexplore.ieee.org/document/9556424
- https://dl.acm.org/doi/pdf/10.1145/3431920.3439302
- https://fabulous.readthedocs.io/en/latest/Usage.html
- https://woset-workshop.github.io/PDFs/2021/a15-slides.pdf
- https://woset-workshop.github.io/PDFs/2021/a15.pdf
-