Bonsai Boosted Decision Tree based Trigger for Hit and Event level selection

Technical Design Report

Vedant Basu[1]

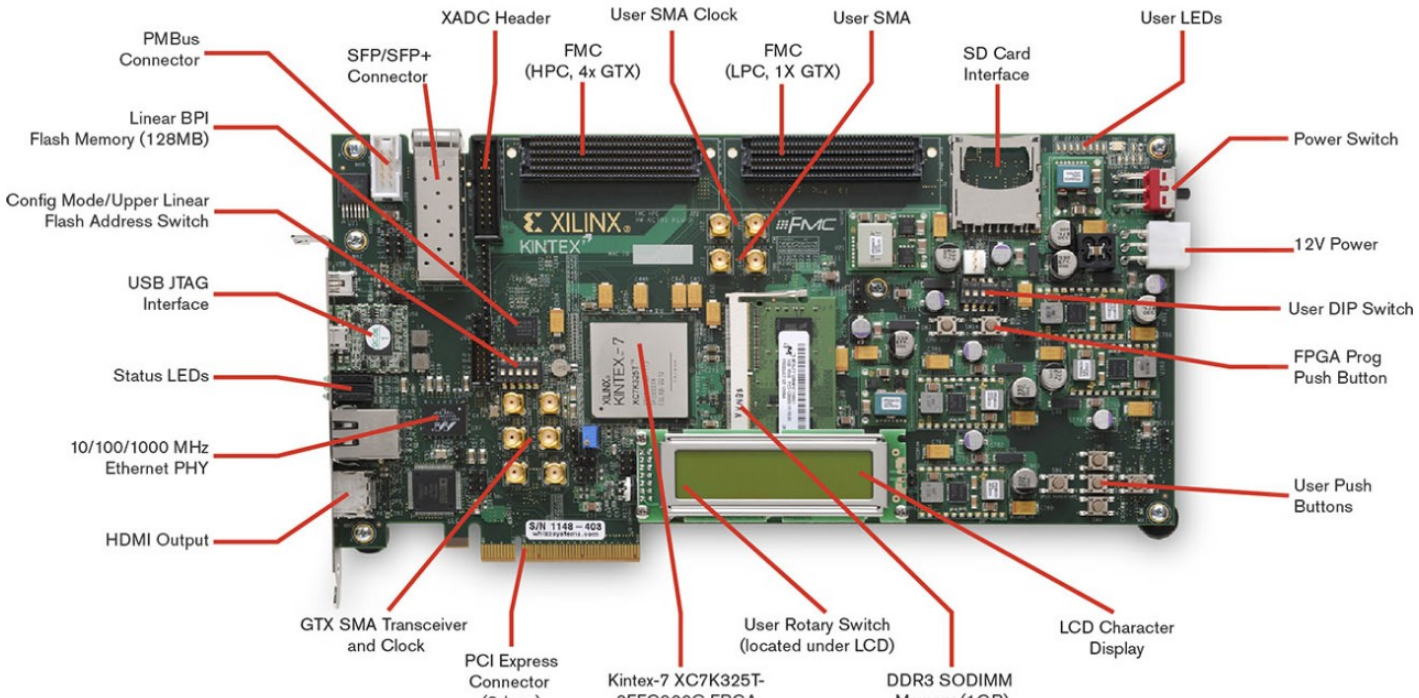
[1]Department of Physics
Indian Institute of Technology Bombay

June-July, 2017

Outline

Problem Statement and Design Considerations

- The objective is to write firmware for the COTRI front-end and motherboard boards, which contain a Xilinx Kintex-7 FPGA chip
- Design and Testing was performed on the KC705 Evaluation Board, at Osaka University during the summer of 2017



Gradient Boosting

- A weak hypothesis or weak learner is defined as one whose performance is at least slightly better than random chance, based on filtering and weighting observations, leaving those observations that the weak learner can handle and focusing on developing new weak learners to handle the remaining difficult observations.[Bro16]
- The idea of boosting came out of the idea of whether a weak learner can be modified to become better, based on additive weighting to optimise a loss function.
- We apply a classification boosted decision tree, using the Energy Deposits on the wire and its neighbours, and the board ID.

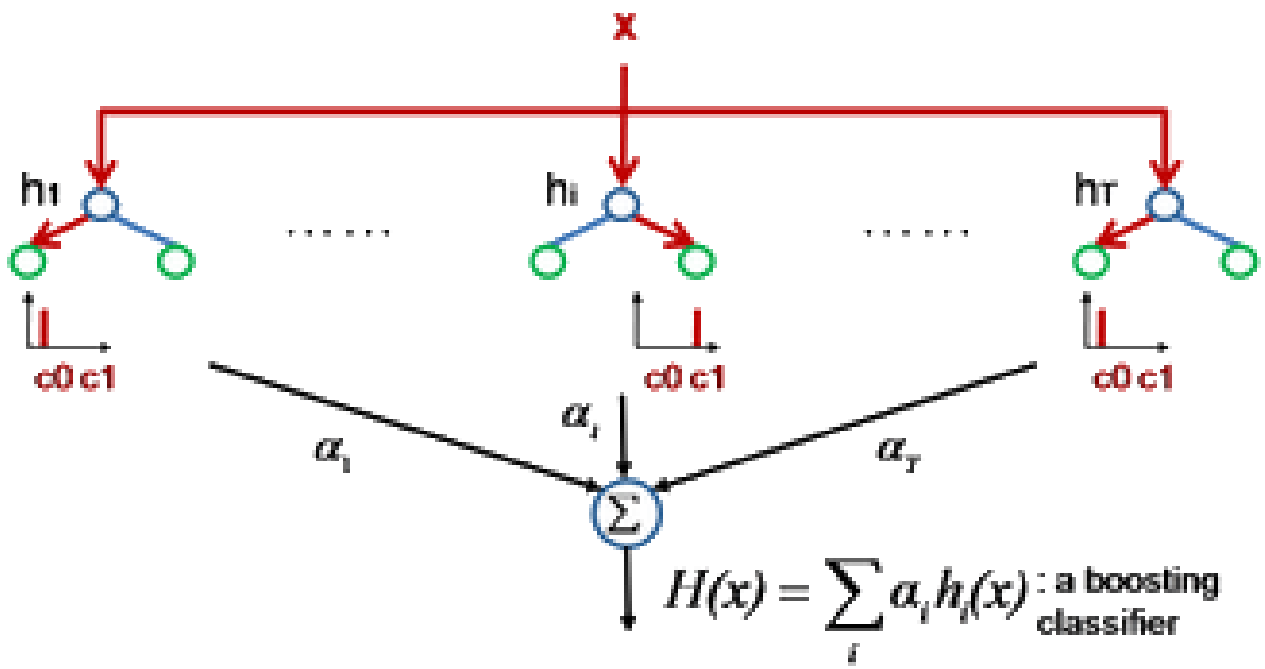


Figure: Gradient Boosting

Bonsai LUT

We train a model using Python's sklearn library, using four bins for our energy deposits.Using the model, we create a lookup table for all possible inputs, which must be stored on board memory. This serves as our hit level scoring criteria

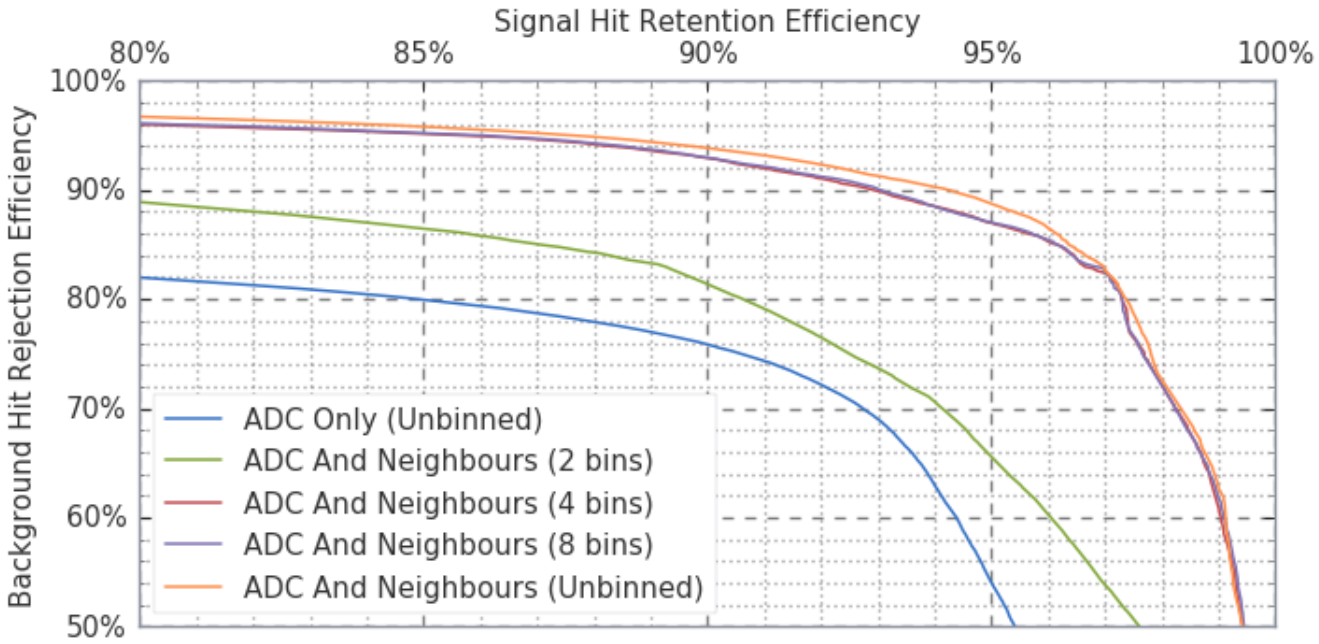


Figure: ROC Curve for different binning schemes

Event Level Trigger

Hough Transform

Using the Hit level scoring criteria, we select good hits which are then used to score the board.We then look for circular arcs connecting the boards which show a large proportion of signal hits, using the Circle Hough Transform, adapted for hardware implementation

Event GBDT

A second Gradient Boosted Tree is used to classify events, which provides a second LUT for processing the Hough Transformed image. However, this is not yet ready for implementation.

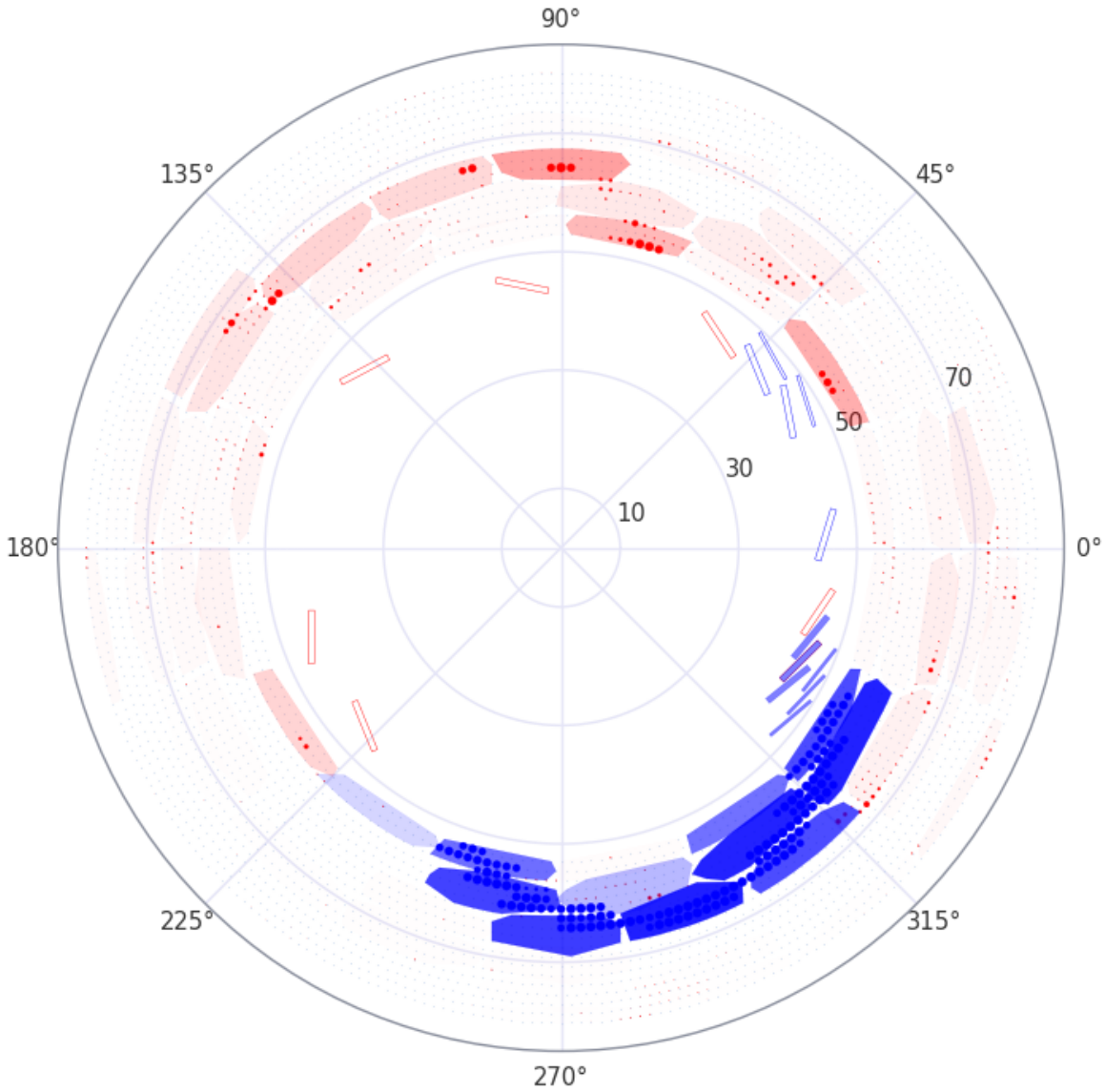


Figure: Result of Hough Transformed Board Array

Design Considerations

- The Bonsai Tree was generated as a Gradient Boosted Decision Tree using scikit-learn and a Look-up Table(LUT) was created to account for all possible combinations. The LUT was stored on the non-volatile off-chip flash, a Micronyx N25Q128 device
- Data was transferred by a UART serial line at 115200 Baud(bits/second). PicoTerm was the terminal software used, with its default settings
- As the data processing was largely sequential in operation, a softcore processor, Xilinx PicoBlaze(KCPSM6) [Cha] , was implemented and the coding was performed in Assembly, rather than Verilog for the main logic.
- To reduce deadtime, a parallel multicore system was also designed, but implementation and testing was incomplete. This system takes cues from the DRS4 Domino Ring Sampler for its logic, using multiple processing threads

Circle Hough Transform

- The Circle Hough Transform was implemented and tested for a 10*10 pixel array. As verilog does not handle the standard accumulating algorithm due to a three dimensional parameter space, a hardware optimized algorithm implementing a one dimensional parameter space was used [XN14]

Data Acquisition Front-End

Data

The software used during testing is PicoTerm, as it is already configured for the required data transfer, and is suitable for large file transfers.

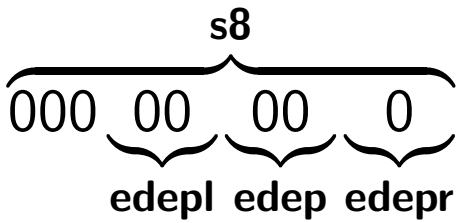
Data Format

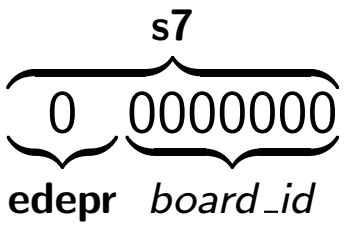
The data is coded into 2 bytes,with the **board_id**(7bits) as the first byte as the first byte **channel_id**(6 bits), binned **energy_deposit** (2 bits) as the second.

Assembly Code-Register Description

- Predefined PicoBlaze macros are used to interface with UART and the SPI Flash, which are contained in *uart_interface_routines.psm* and *N25Q128_SPI_routines.psm*
- The Flash is addressed with 24 bits, split into 3 8bit registers, **s9** (sector) , **s8** (Page) , **s7** (byte). **s9** is set to a constant depending on the function to be performed
- Using the macros **write_spi_byte** or **read_spi_byte** require the data to be loaded into register **s2**, along with the aforementioned addressing
- when reading 2bit hex values from memory, it is advisable to load the value into **s4**, and call the **send_hex_byte** macro if using UART
- The UART communication is handled by **UART_TX** and **UART_RX** macros, which are interfaced with **s5**
- In addition, sA is usually used as a looping variable.

LOOKUP TABLES-BBDT

- The BBDT LUT (104*64) is stored in sector 163'd, indexed with 2bit energy deposits (**edep**) of the wire and the values of the adjacent wires, along with the **board_id**, 7bits The address format is 13 bits stored across **s7**, **s8** (**edepl**,**edep**,**edepr**,**brd_id**).
- 

- 

- The BBDT LUT is entered into a file *"picoreadfile.txt"* in the same directory as PicoTerm, and is automatically read by the program when the W option is selected in the terminal
- Before writing to any memory, it is prudent to erase the sector, calling macro **erase_spi_sector**. The function E on terminal also performs the same function
- As the Board/Channel IDs are mismatched with Layer/Cell IDs, a channel map is implemented as another LUT, this time stored
- Tested and Fully Operational

LOOKUP TABLES-CHANMAP

- As the Board/Channel and Layer/Cell are not in order, the input values must be sorted according to Cell ID, following which they can be processed
- This lookup table is indexed by a 7bit board id, and a 6-bit channel id. The Layer and Cell IDs are 5 and 9 bits respectively. This necessitates the use of two bytes for each element of this Table
- For easy indexing, these two bytes are stored in different sectors, with LSB in 161 and the remaining 6 bits in 160.
- Tested. Data Transfer to FLASH stable, although sometimes glitches at end of file.

Data Processing

- The data input is taken to be 4 bit hexadecimal, in format **board_id**, **channel_id**, and binned **edep**. It is then stored into sectors 164 and 165, according to cell ID
- This data then undergoes processing by the BBDT LUT
- The values are summed for every channel on a board, and stored as a two byte value in sector 162. As there are only 104 boards, the two bytes are stored on the same page, separated by 128 spaces
- Data is output to a timestamped output file in the PicoTerm folder
- This has not been fully tested yet, due to lack of simulated raw data

Classical Circle Hough Transform

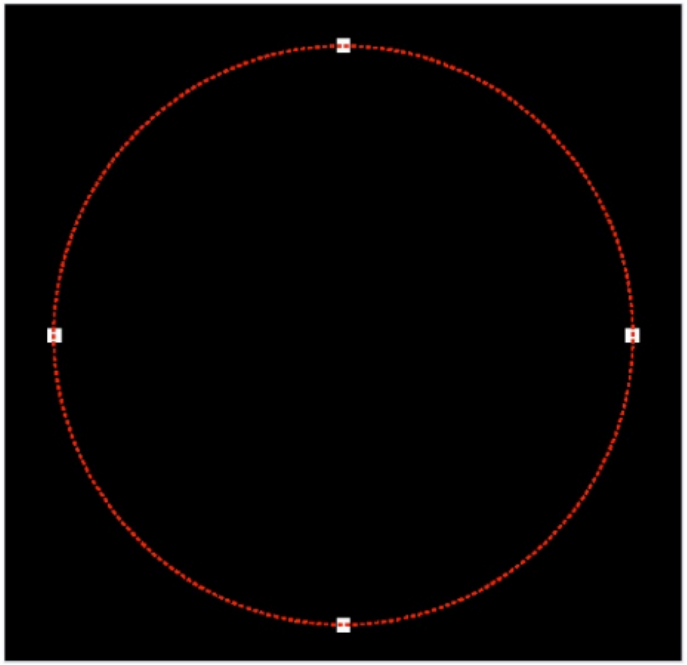
Introduction

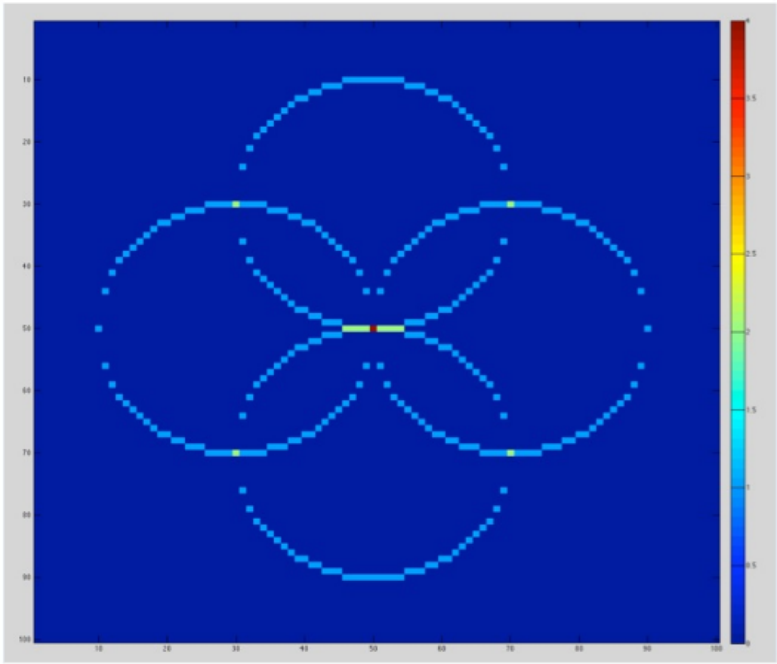
The circle Hough Transform is an image processing algorithm used to detect Arcs and circles in a pixel array. This is useful as we can classify events based on suitability of tracks, and thus improve trigger efficiency

Algorithm

The classical Circle Hough Transform uses an accumulator array three dimensional parameter space, spanned by the centre coordinates and the radius. This was implemented as a testing ground, but was unsuitable for scaling up, and so this approach was abandoned

Algorithm





Consider 4 points on a circle in the original image (left). The circle Hough transform is shown in the right. Note that the radius is assumed to be known. For each (x,y) of the four points (white points) in the original image, it can define a circle in the Hough parameter space centered at (x, y) with radius r. An accumulator matrix is used for tracking the intersection point. In the parameter space, the voting number of points through which the circle passing would be increased by one. Then the local maxima point (the red point in the center in the right figure) can be found. The position (a, b) of the maxima would be the center of the original circle.

[Cir17]

One Dimensional Hough Transform

Implementation

For a hardware implementation, we use an algorithm [XN14] which stores the centre coordinates in a one dimensional array. We then calculate the radius, and accumulate the values. As a preliminary check, we could apply a simple threshold cut, but it is certainly possible to refine our transform after we have detected circles

Optimisation

One of the advantages is the use of pipelining and DSP slices. To find the radius, we use a Xilinx IP Core, which uses the CORDIC Iterative Algorithm to efficiently calculate square roots with a Minimum Area and Maximum Pipelining optimisation. However, this requires further testing to optimize deadtime [Xil]

Algorithm

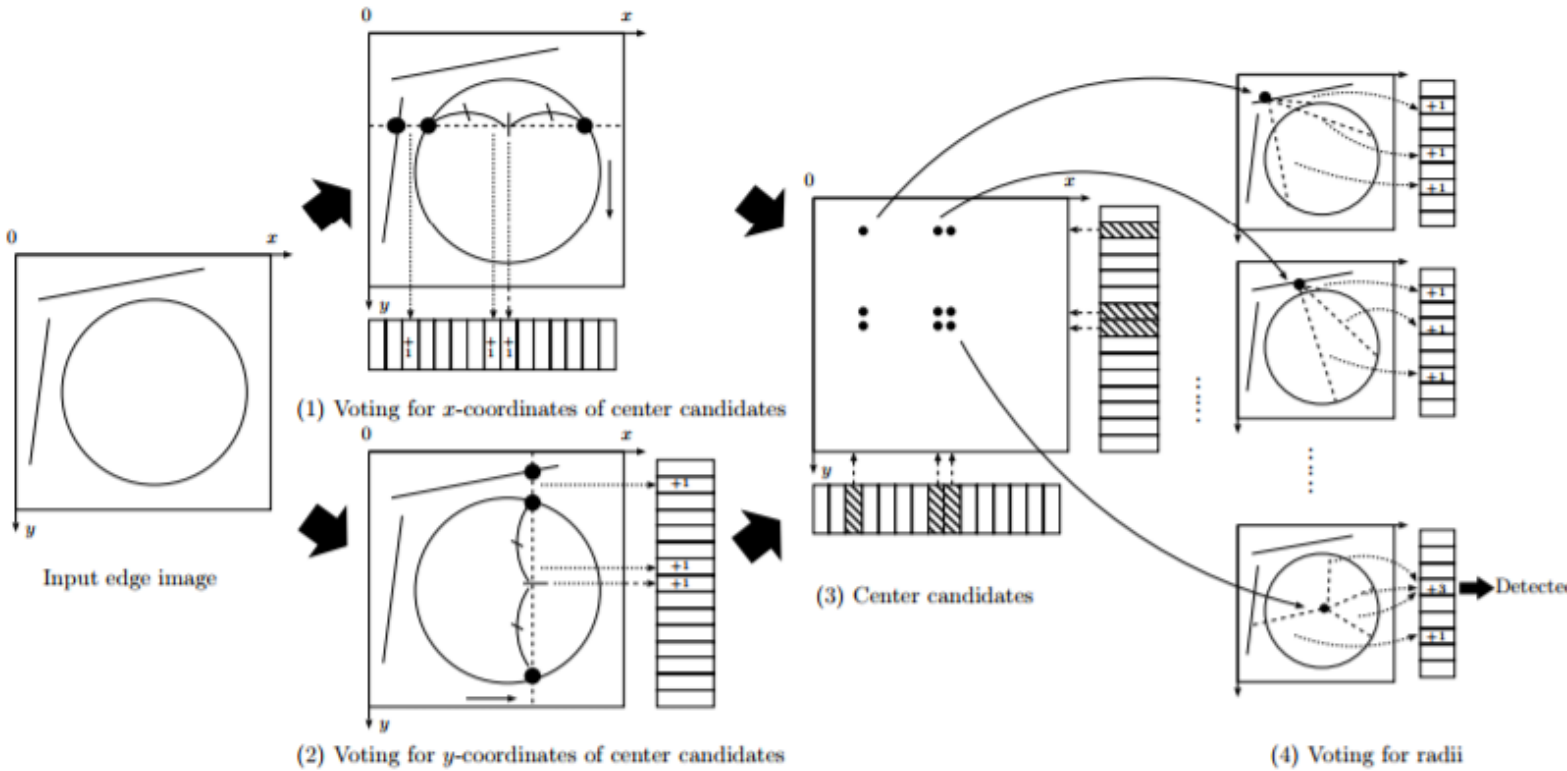


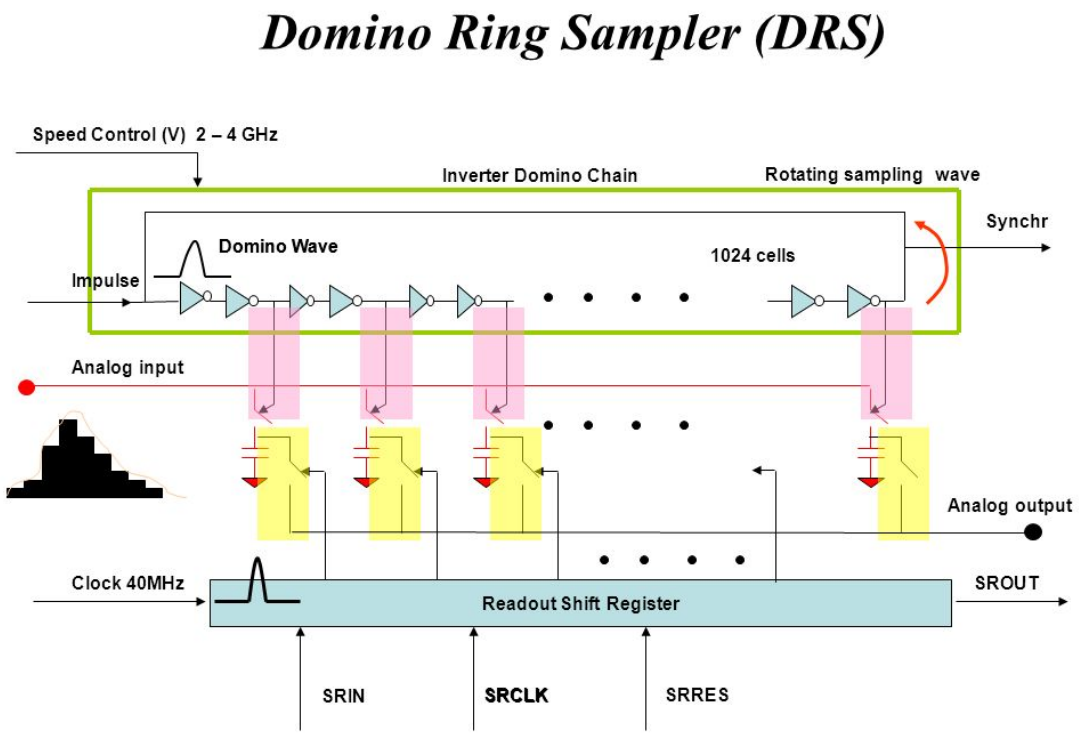
Figure 2. The outline of the one-dimensional Hough transform algorithm for circle detection [XN14]

Implementation

Block RAMs and DSP slices have not been explicitly instantiated, and this is a further optimisation which can be performed. A pair of arrays, **x_hit** and **y_hit** are used to store the edge points,to which we try to fit the circles. A 10*10 array was used to test the implementation, however, once finalized, this will have to be modified to account for the orientation of the RECBE boards in the final CyDet. Further testing is required, along with proper storage of centre and radius values.

Domino Ring Sampler

A Domino Ring Sampler is a device which allows for sampling of a high frequency input signal into multiple parallelised channels, which are serially sampled. This reduces deadtime and increases detector latency.



Application

If a single core is responsible for all Data input and output transactions, it can then transfer data into multiple identical cores which can then perform the sequential processing algorithm

Implementation

This is still incomplete, although preliminary interfacing with twin cores has proved successful with pipeline buffers

Use of Block RAMs and DSP slices

Optimisation

In the final implementation, it is advisable to explicitly instantiate Block Rams to store the various matrices, and the DSP48 slice provided by Xilinx may give greater performance


Yasuaki Ito Xin Zhou and Koji Nakano. "An Efficient Implementation of the One-Dimensional Hough Transform Algorithm for Circle Detection on the FPGA". In: (2014).


Jason Brownlee. *A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning*. 2016. URL: http://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/.


Circle Hough Transform. *Circle Hough Transform Wikipedia, The Free Encyclopedia*. 2017. URL: https://en.wikipedia.org/wiki/Circle_Hough_Transform.


Ken Chapman. "Xilinx KCPSM6 PicoBlaze". In: ().

Xilinx. "CORDIC v6.0 LogiCore User Guide PG105". In: ().