

Part B

An advanced scalable hardware accelerator for optical flow

An Advanced Scalable Hardware Accelerator for Optical Flow

An advanced scalable hardware accelerator for optical flow

Chen Dahan

313428468

An advanced scalable hardware accelerator for optical flow

Ofri Bata

206459612

An advanced scalable hardware accelerator for optical flow

Shahar Gino

22/12/2024

**Acknowledgments**

We would like to express my sincere gratitude to all those who have contributed to the successful completion of this project.

First, our supervisor, Shaher Gino, for his invaluable support, guidance, and encouragement throughout the project. And for the time and dedication he invested in helping us.

We would also like to extend our thanks to the entire staff of the VLSI Lab, Goel Samuel and Riyad Nasrallah for their assistance during the development of this project.

**Table of Contents**

[1.Introduction 10](#_Toc177907818)

[1.1. Optical-Flow background 10](#_Toc177907819)

[1.2. Object-Detection and Video-Registration 13](#_Toc177907820)

[1.3. AMBA APB 15](#_Toc177907821)

[1.4. Paper study 21](#_Toc177907822)

[2. Accelerator Implementation 30](#_Toc177907823)

[2.1. Block diagram 30](#_Toc177907824)

[2.2. Block description (+flow-chart) 31](#_Toc177907825)

[2.3. Pins description 32](#_Toc177907826)

[2.4. Clocks and Resets 33](#_Toc177907827)

[2.5. Interfaces description 33](#_Toc177907828)

[2.6. Sub-units description 34](#_Toc177907829)

[2.6.1 oflow\_core 34](#_Toc177907830)

[2.6.2 oflow\_registration 38](#_Toc177907831)

[2.6.3 oflow\_conflict\_resolve 42](#_Toc177907832)

[2.6.4 oflow\_Mem\_buffer\_wrapper 49](#_Toc177907833)

[2.7. Performance 66](#_Toc177907834)

[2.8. Synthesis 66](#_Toc177907834)

[2.8.1 Technology and Constrains (SDC) 34](#_Toc177907830)

[2.8.2 FloorPlan 38](#_Toc177907831)

[2.8.3 Area and StaticPower 42](#_Toc177907832)

[2.8.3 Clock Tree Synthesis (CTS) 42](#_Toc177907832)

[2.9. Programmer’s Guide 66](#_Toc177907834)

[3. Zero-order ("aliveness") verification 67](#_Toc177907835)

[3.1. Test-Plan and Test-Results oflow\_top 67](#_Toc177907836)

[3.2. Test-Plan and Test-Results oflow\_core 69](#_Toc177907837)

[3.3. Test-Plan and Test-Results oflow\_registration 72](#_Toc177907838)

[3.4. Test-Plan and Test-Results oflow\_conflict\_resolve 74](#_Toc177907839)

[3.5. Test-Plan and Test-Results oflow\_mem\_buffer\_wrapper 74](#_Toc177907839)

[4. Summary 85](#_Toc177907840)

[4.1 Project's summary 85](#_Toc177907841)

[4.2 Take message home 86](#_Toc177907842)

[4.3 Next steps 86](#_Toc177907843)

[5.References 87](#_Toc177907844)

**Table of Figures**

[Figure 1: Optical Flow Illustration 12](#_Toc179729934)

[Figure 2: BBoxes after object detection 14](#_Toc179729935)

[Figure 3: APB Protocol Interface 16](#_Toc179729936)

[Figure 4: APB - Write transfer with no wait states 17](#_Toc179729937)

[Figure 5: APB - Connection between Master and Slave 18](#_Toc179729938)

[Figure 6: APB - Read transfer with no wait states 19](#_Toc179729939)

[Figure 7: Paper Study - System Architecture 25](#_Toc179729940)

[Figure 8: Paper Study - Finite State Machine 25](#_Toc179729941)

[Figure 9: Paper Study - The relationship between the block radius and AAE 27](#_Toc179729942)

[Figure 10:Top level-external interface 30](#_Toc179729943)

[Figure 11:Top level-block diagram 30](#_Toc179729944)

[Figure 12: Oflow-Core - block diagram 34](#_Toc179729945)

[Figure 13: oflow-pe - block diagram 38](#_Toc179729946)

[Figure 14: oflow-Feature-Extraction - block diagram 43](#_Toc179729947)

[Figure 15: oflow-Feature-Extraction - FSM 44](#_Toc179729948)

[Figure 16: oflow-Mem-Buffer-Wrapper - block diagram 47](#_Toc179729949)

[Figure 17: oflow-Mem-Buffer All-Mem - block diagram 51](#_Toc179729950)

[Figure 18: oflow-Similarity-Metric - block diagram 54](#_Toc179729951)

[Figure 19: oflow-Similarity-Metric - Internal block diagram 56](#_Toc179729952)

[Figure 20: oflow-Similarity-Metric - FSM 57](#_Toc179729953)

[Figure 21: oflow-calc-iou - Block Diagram 63](#_Toc179729954)

[Figure 22: Calc-Iou - IoU Examples 70](#_Toc179729955)

[Figure 23: oflow-Calc-Iou - IoU formula 70](#_Toc179729956)

[Figure 24: Test Results - Reg-File - Write transaction 72](#_Toc179729957)

[Figure 25: Test Results - Reg-File - Read transaction 73](#_Toc179729958)

[Figure 26: Test result - Feature-Extraction 75](#_Toc179729959)

[Figure 27: Test result – oflow-Mem-Buffer - Write transaction 77](#_Toc179729960)

[Figure 28: Test result – oflow-Mem-Buffer - Read transaction 78](#_Toc179729961)

[Figure 29: Test result – oflow-Similarity-Metric - Overlap case - Inputs 79](#_Toc179729962)

[Figure 30: Test result – oflow-Similarity-Metric - Overlap case - Outputs 79](#_Toc179729963)

[Figure 31: Test result – oflow-Similarity-Metric -Without overlap case - Inputs 83](#_Toc179729964)

[Figure 32: Test result - Similarity-Metric -Without overlap case - Outputs 83](#_Toc179729965)

[Figure 33: Test result – oflow-Similarity-Metric - calc iou- With overlap case 86](#_Toc179729966)

[Figure 34: Test result – oflow-Similarity-Metric -calc iou- Without overlap case 88](#_Toc179729967)

**Table of Tables**

[Table 1: Conventions & Terms 7](#_Toc177907881)

[Table 2: Paper Study - Data sequence and Algorithm 23](#_Toc177907882)

[Table 3: Paper Study - Experimental Results 25](#_Toc177907883)

[Table 4: Paper Study - Time Complexity Analysis 27](#_Toc177907884)

[Table 5: Oflow-Top - Pins Description 32](#_Toc177907885)

[Table 6: Oflow-Reg-File - Pins Description 36](#_Toc177907886)

[Table 7: Oflow-Feature-Extraction - Pins Description 39](#_Toc177907887)

[Table 8: Oflow-Mem-Buffer – Structure of the desired memory 44](#_Toc177907888)

[Table 9: Oflow-Mem-Buffer - Pins Description 46](#_Toc177907889)

[Table 10: Oflow-Similarity-Metric - Pins Description 55](#_Toc177907890)

[Table 11: Oflow-Feature-Extraction - Test Plan 68](#_Toc177907891)

[Table 12: Oflow-Similarity-Metric with overlap - Test Plan 76](#_Toc177907892)

[Table 13: Oflow-Similarity-Metric without overlap - Test Plan 78](#_Toc177907893)

[Table 14: Oflow-Calc-Iou with overlap - Test Plan 81](#_Toc177907894)

[Table 15: Oflow-Calc-Iou without overlap - Test Plan 83](#_Toc177907895)

**Conventions & Terms**

|  |  |
| --- | --- |
| **Term** | **Description** |
| CM | Center of Mass |
| TL | Top Left |
| BR | Bottom Right |
| BBOX | Bounding Box |
| IoU | Intersection Over Union |
| WO | With Out |
| PE | Proccess engine |
| OF, oflow | Optical Flow |
| LUT | Look-Up Table |
| CR | Conflict Resolve |

Table 1: Conventions & Terms

**Abstract**

Today the demand for object registration is growing, for example: security cameras and games with multiple players (like soccer).

Software solutions like Byte-Track are not fast enough for this, therefore the request for hardware accelerator increases.

This project deals with this demand by scalable hardware, that performs the object registration in real time, based on object detection that is implemented by external units.

This accelerator has an original and innovative algorithm, especially the conflict resolve unit.

The algorithm divides into 4 main steps. In the first step a DMA send the BBoxes to the feature extraction unit. In this step, new features are calculated for the next steps. The next step is registration unit, this step is responsible for similarity calculation and labeling the BBoxes. In the next stage the conflicts are resolved by unit which is called conflict resolve. In the last step the features and IDs of the BBoxes in the current frame are stored in the memory for further comparison and calculation of the future frames.

# 1.Introduction

**Major Contribution**:

Original and new HW algorithm for the object-registration requirement, especially Conflict Resolve unit.

## 1.1. Optical-Flow background

**definition:**

Optical Flow pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer and a scene.

Optical flow is a powerful tool that can be used to solve a variety of problems in computer vision, including:

* Motion tracking
* Video stabilization
* Structure from motion
* Video compression
* Video object segmentation, etc.

**Estimation:**

In order to estimate motion as either instantaneous image velocities or discrete image displacements, there is a need for a consistent series of images.

there are various methods to calculate the motion between two frames of scene, which are taken at times, t and .

some of the methods are based gradient.

These methods are called differential since they are based on local [Taylor series](https://en.wikipedia.org/wiki/Taylor_series) approximations of the image signal.

for the 2D case:

A voxel at location(x, y, t) with intensity I(x, y, t) will have moved by between 2 frames.

Optical flow works on several assumptions:

1. The pixel intensities of an object do not change between consecutive frames.
2. Neighboring pixels have similar motion.

, with Tylor series:

The next step is to perform linearization; thus, the calculation of the operation point is:

�(�,�,�)=�(�+Δ�,�+Δ�,�+Δ�)

after dividing the equation by the result is:

And after that, re-arrange the equation:

The above equation is called Optical Flow equation. From this equation, Ix and Iy can be found, they are image gradients. Similarly,  is the gradient along time. But () is unknown. This equation cannot be solved with two unknown variables. So, several methods are provided to solve this problem.

A line of red dots

Description automatically generated

Figure 1: Optical Flow Illustration

This figure shows a ball moving in 5 consecutive frames. The arrow shows its displacement vector.

## 1.2. Object-Detection and Video-Registration

**definition**:

**Object-Detection:** Object detection is a computer vision technique for locating instances of objects in images or videos. Object detection algorithms typically leverage machine learning or deep learning to produce meaningful results. It is an important part of many applications, such as surveillance, self-driving cars, or robotics.

**Video-Registration:**

Video-registration is matching the detected instances of objects through the consequent frames in scene. keeping track of each detected objects uniquely among the incoming frames.

**YOLO**:

One of the leading algorithms of object detection is: YOLO (you only look once).

This is a machine learning algorithm. YOLO proposes using an end-to-end fully convolutional neural network (CNN) to process an image by making predictions of bounding boxes and class probabilities all at once. It differs from the approach taken by previous object detection algorithms.

YOLO is a single-shot detector, Single-shot object detection uses a single pass of the input image to make predictions about the presence and location of objects in the image. It processes an entire image in a single pass, making them computationally efficient.

However, single-shot object detection is generally less accurate than other methods, and it’s less effective in detecting small objects.

 YOLO achieved state-of-the-art results, beating other real-time object detection algorithms by a large margin.

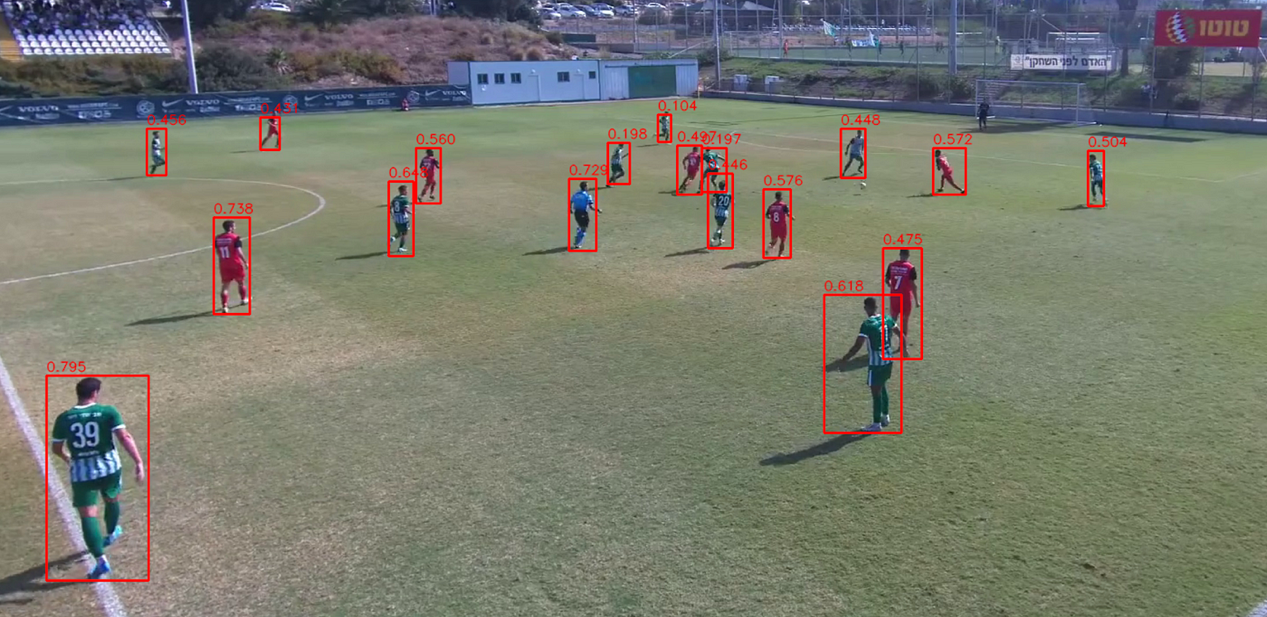


Figure 2: BBoxes after object detection

Figure of bounding box around the object that created by YOLO algorithm.

**The connection to optical flow:**

YOLO can be used in conjunction with optical flow to improve the accuracy of object tracking. Optical flow can be used to track the movement of objects between frames, and this information can be used to improve the accuracy of YOLO's object detections

In objects registration, optical flow can be used to register two images of the same scene taken from different viewpoints. This is done by estimating the optical flow between the two images. The optical flow vectors can then be used to warp one image to match the other image**.**

## 1.3. AMBA APB

Advanced Microcontroller Bus Architecture (AMBA) Advanced Peripheral Bus (APB) Protocol Specification

**definition**:

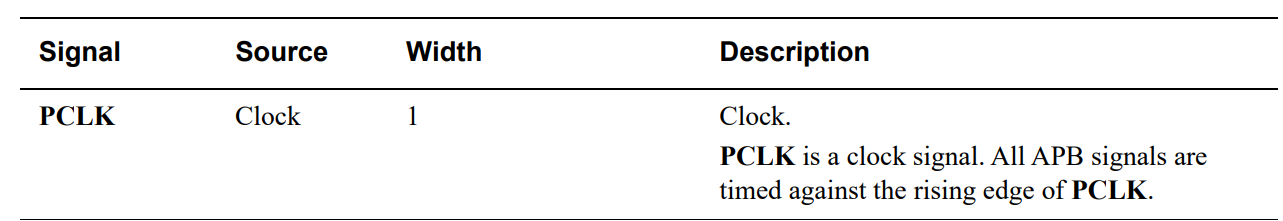
The APB protocol is a low-cost interface, optimized for minimal power consumption and reduced interface complexity. The APB interface is not pipelined and is a simple, synchronous protocol. Every transfer takes at least two cycles to complete.

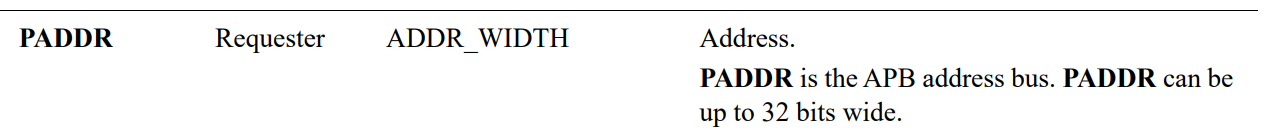
In this project, CPU needs to access the RegFile which resides in the hardware accelerator module. Thus, it will use the APB protocol which is built for these cases.

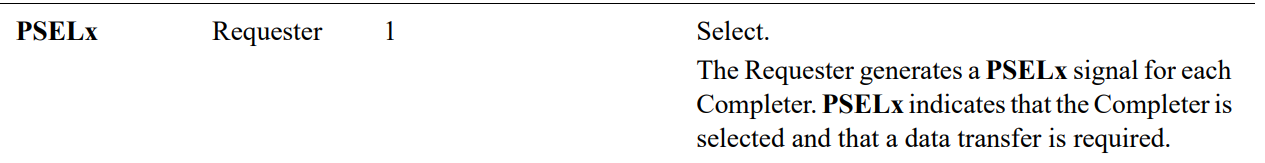
APB peripherals are typically connected to the main memory system using an APB bridge. For example, a bridge from AXI to APB could be used to connect several APB peripherals to an AXI memory system.

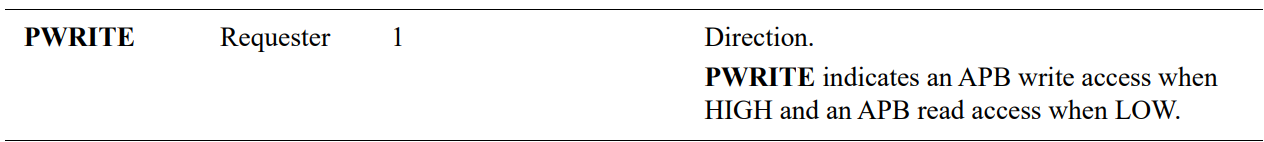
APB transfers are initiated by an APB bridge. APB bridges can also be referred to as a Requester. A peripheral interface responds to requests. APB peripherals can also be referred to as Completer. This specification will use Requester and Completer.

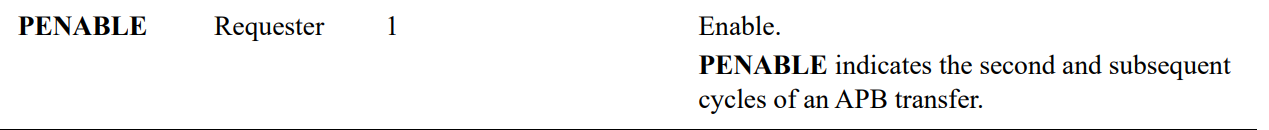
**Signals:**

****

****

****

****

****

**A close-up of a white background

Description automatically generated** ****

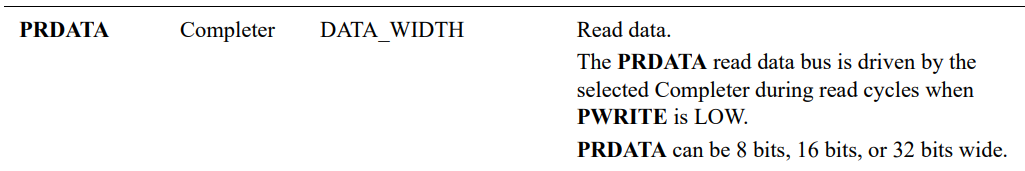
****

Figure 3: APB Protocol Interface

**Write Transfer:**

Write transfers -This section describes the following types of write transfer:

• With no wait states

• With wait states

All signals shown in this section are sampled at the rising edge of PCLK

**with no wait states**

**A diagram of a diagram

Description automatically generated**

Figure 4: APB - Write transfer with no wait states

There are 3 main phases for write transfer**:**

* **The Setup phase** of the write transfer occurs at T1 in Figure 4. The select signal, PSEL, is asserted, which means that PADDR, PWRITE ('1'), and PWDATA must be valid.
* **The Access phase** of the write transfer is shown at T2 in Figure 4 where PENABLE is asserted. PREADY is asserted by the Completer (Slave) at the rising edge of PCLK to indicate that the write data will be accepted at T3. PADDR, PWDATA, and any other control signals must be stable until the transfer is complete.
* **T3-the writing stage**, the data will be written to the Slave register.

At the end of the transfer, PENABLE is de-asserted. PSEL is also de-asserted, unless there is another transfer to the same peripheral.

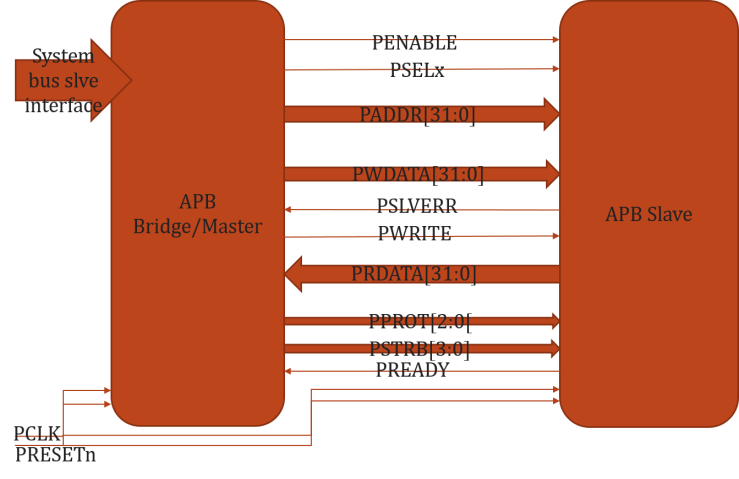


Figure 5: APB - Connection between Master and Slave

**Read transfer:**

Two types of read transfer are described in this section:

• With no wait states

• With wait states

All signals shown in this section are sampled at the rising edge of PCLK. 3.3.1

**With no wait states**

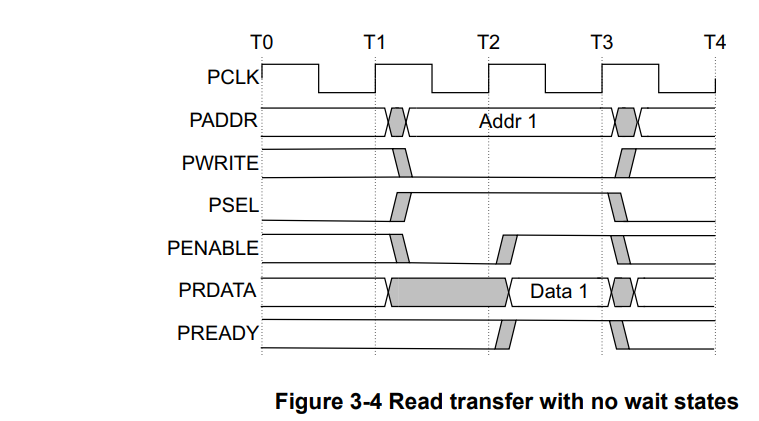
****

Figure 6: APB - Read transfer with no wait states

* **The Setup phase** of the read transfer occurs at T1 in Figure 6. The select signal, PSEL, is asserted, which means that PADDR, PWRITE ('0') must be valid.
* **The Access phase** of the read transfer is shown at T2 in Figure 6 where PENABLE is asserted. PREADY is asserted by the Completer (Slave) at the rising edge of PCLK to indicate that the read data will be accepted at T3. PADDR, PRDATA, and any other control signals must be stable until the transfer completes.
* **T3-the reading stage**, the data will be read to the master.

At the end of the transfer, PENABLE is de-asserted. PSEL is also de-asserted, unless there is another transfer to the same peripheral.

The Completer must provide the data before the end of the read transfer

## 1.4. Paper study

**introduction**:

The paper study proposes an event-based block matching algorithm to calculate OF (Optical Flow) on FPGA. In computer vision, OF describes the motion field induced by camera movement through space. OF methods is a search over possible flows to select the most likely one at each image or feature location. This search with dense image blocks is expensive and difficult to calculate on an embedded platform in real time

The DVS (Dynamic Vision Sensor) is data-driven rather than regular-sample driven. Regular-sample driven means camera sends the output data at a fixed interval; thus, it is called a frame-based camera. However, the DVS output is driven by brightness changes rather than a fixed sample interval. Therefore, new OF methods need to be designed. There are various methods that offered comparable accuracy for sharp and sparse edges, but all fail on textured or low spatial frequency inputs, because the underlying assumptions (e.g. smooth gradients or isolated edges) are violated.

In video technology, OF is called motion estimation (ME). The pipeline for ME includes block matching. Block matching means that rectangular blocks of pixels are matched between frames to find the best match. Block matching is computationally expensive. Therefore, it is recommended to use hardware instead of software, as implemented in this experiment.

**Propose Method**:

The output of DVS is a stream of brightness change events. Each event has:

* microsecond timestamp
* a pixel address
* a binary polarity describing the sign of the brightness change

In this experiment, events are accumulated into time slice frames as binary bitmap frames called slices.

A block is a square centered around the incoming event’s location. Matching is based on a distance metric. In this experiment, they use Hamming Distance (HD). HD is the count of the number of differing bits. For bitmaps, HD is exactly like SAD (Sum-of-Absolute-Differences).

The hardware evaluation system is divided into two parts, one for data sequencing and monitoring and the other for the algorithm implementation.

|  |  |  |
| --- | --- | --- |
| **data sequencing and monitoring** | | **algorithm implementation** |
| Sequencer | Monitor | When event arrives, the next things are chosen:   * on centered block as a reference in slice t-d * 9 blocks in slice t-2d   They will be sent to the HD module to calculate the distance.  The algorithm finds the most similar block on the t-2d slice. According to the brightness-constancy assumption of OF, A similar block in the t-2d slice should be seen for the block that best matches the actual OF. |
| converts the event-based dataset into real-time hardware events sent to the OF FPGA | collects the OF events and sends them over USB to jAER for rendering and analysis | 1. Hamming Distance: The implementation of one HD block is shown in Fig 3. A total of 81 XOR logic gates receive input from corresponding pixels on the slices. The XOR outputs are summed to compute the HD. 2. Minimum Distance Computation: The last step of the algorithm is to find the minimum distance candidate. Part of the novel minimum circuit is shown in Fig 4. It is a parallel implementation that outputs the index of the minimum distance direction. At the end, the part whose sum is zero is the minimum candidate. |
| The target of this stage is to compare software and hardware processing of OF algorithm | | The minimum distance candidate is determined in one clock cycle compared to SW implementation. |

Table 2: Paper Study - Data sequence and Algorithm

The OF architecture (Figure 7) consists of 3 main modules: Finite State Machine (FSM), Slice RAMs and Rotation Control Logic.

FSM consists of 3 main stages: data receiving module, OF calculation module, and data sending module.

A diagram of a computer system

Description automatically generatedA diagram of a machine

Description automatically generated

Figure 7: Paper Study - System Architecture

One is the current collecting slice starting at time t and the other two are the past two slices starting at times t-d and t-2d. d is the slice duration. At intervals of d, the rotation control logic rotates the three slices. The t slice accumulates new data. It starts out emptyotation control logic rotates the three slices. The t slice accumulates new data. It starts out emptyand gradually accumulates events, so it cannot be used for matching to past slices. The two past slices are used for OF, but the OF computation is done at the location of each event stored into the t slice, and thus is driven by these events.

t-2d slice

t-d slice

t slice

Three 240x180-pixel DVS event bitmaps slices

Figure 8: Paper Study - Finite State Machine

**Experimental Results:**

In the experiment, three different sample datasets that are real DVS were tested. They are listed in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Dataset** | **Characterization** | **How was it recorded?** | **Result** |
| Box Translation | edges | the boxes scene has a box in the foreground and clutter in the background and the camera pans to the left | most vectors point correctly east |
| Pavement | sparse points | the camera was down looking and carried by hand | most vectors point southeast |
| Gravel | dense textures | dataset is recorded outside.  movement is eastward | most vectors point east |

Table 3: Paper Study - Experimental Results

Errors are mostly caused by DVS noise or aperture ambiguity for the extended edges.

1. **Accuracy analysis**

two ways to calculate event based OF accuracy:

1. AEE - measures error in the direction of estimated flow includes speed error.
2. AAE - measures error in the direction of estimated flow.

They chose two other algorithms to compare with their algorithm: Lucas-Kanade and Local Plane.

It can be seen that the block matching algorithm has the best accuracy for AEE and second-best for AAE.

In addition, they show in Fig. 6 that bigger block dimension leads to better accuracy. However, larger blocks consume more logic and reduce spatial resolution of the flow.

A graph with a line

Description automatically generated

Figure 9: Paper Study - The relationship between the block radius and AAE

1. **Time complexity analysis**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Software Implementation | | | FPGA Implementation | | |
| Complexity | Quadratically | | | Linearly | | |
| Processing Part | **reading data from three slices** | **HD calculation** | **looking for the minimum** | **reading data from three slices** | **HD calculation** | **looking for the minimum** |
| Time | linear time to read data from RAM since multiple data cannot be read from one RAM simultaneously. | quadratic | quadratic | linear time to read data from RAM since multiple data cannot be read from one RAM simultaneously. | 1 clock cycle | 1 clock cycle |
| Summary | 4.5 | | | (block dimension + 2) cycles = 0.22 | | |

Table 4: Paper Study - Time Complexity Analysis

**Conclusion**:

In this paper, they invented a new method to estimate the event-based optical flow on FPGA in real time. The software computational cost of Hamming Distance increases quadratically as the block size increases, however, in FPGA, all the bits in the block can be calculated at the same time which leads to a constant time for all block sizes. This greatly reduces the overall computation time for the FPGA implementation, which is 20 times faster than the software implementation.

There are additional possible improvements:

* Considering the speed also
* Explore other distance metrics
* Implement feedback control on the slice duration to better exploit the unique feature of DVS event output that it can be processed at any desired sample rate

# 2. Accelerator Implementation

## 2.1. Block diagram

**oflow\_top overview**

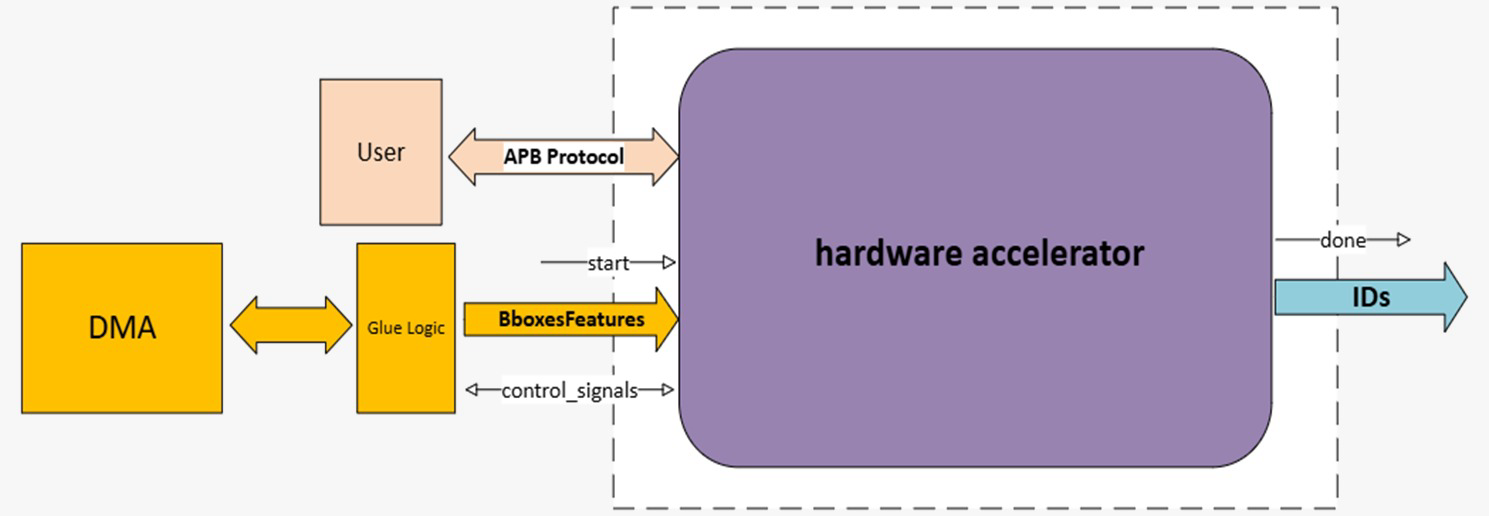
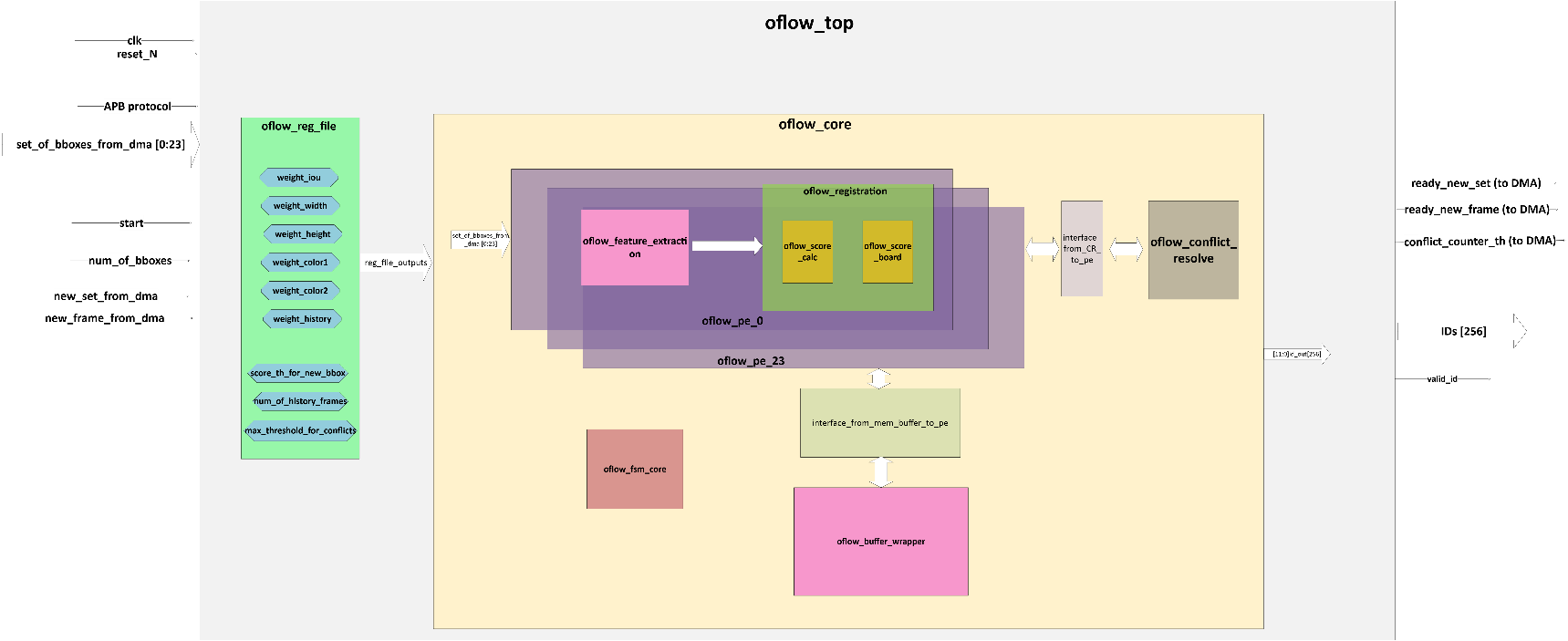
****

Figure 10:Top level-external interface

Figure 11:Top level-block diagram

****

## 2.2. Block description (+flow-chart)

The oflow\_top unit is responsible for processing data received from the GLU logic, specifically handling a set of 24 bounding boxes (BBoxes) for each video frame. The primary function of this unit is to label these bounding boxes on a frame-by-frame basis.

1. **Bounding Box Identification:**
   * **New Bounding Boxes:** For any BBox that is newly identified in a frame (its score is above threshold value), the unit will assign a unique ID.
   * **Existing Bounding Boxes:** If a BBox has been identified in previous frames, the unit will recognize it and assign the same ID that was previously given, ensuring consistent tracking across frames.
2. **Conflict Resolve:**
   * In scenarios where more than 10 BBoxes are erroneously recognized as having the same ID (conflict), the unit will signal the DMA that it had 10 conflicts and reset the process.
   * And if there is a conflict between more than 1 BBox this unit will resolve this by assigning the desired ID to the BBox with the minimum score, which is considered the most appropriate match.
3. **Communication and Control:**
   * The unit interfaces with the user via the Advanced Peripheral Bus (APB). Through this interface, the user can set the weighting for each similarity metric used in BBox identification and can set the score threshold for new BBox and the number of history frames.
   * The unit will signal the GLU logic when it is ready to receive a new set of bounding boxes or a new frame for processing.
   * Upon completion of processing, the unit will publish the final IDs assigned to each BBox.

This unit consists of 2 units: oflow\_reg\_file and oflow\_core which will be explored later.

## 2.3. Pins description

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal\_Name** | **Type** | **Width** | **Description** |
| apb\_control | I/O |  | Ports of apb protocol to connect between user (Master) and the accelerator (slave) |
| start | I | 1 | Indication to start a new flow of frames |
| set\_of\_BBoxes\_ from\_dma[23:0] | I | 86 | Array of BBoxes' features, contains: [x, y, width, height, color1, color2] |
| num\_of\_BBoxes | I | 8 | Total number of the BBoxes in the current frame |
| new\_set\_from\_dma | I | 1 | Indication that new set of the frame is ready |
| new\_frame\_from\_dma | I | 1 | Indication that new frame is ready |
| ready\_new\_set | O | 1 | Indication to DMA (glue logic) that the proccess of the current set in the frame is done, and the accelerator is ready for next set |
| ready\_new\_frame | O | 1 | Indication to DMA (glue logic) that the proccess of the current frame is done and the labels are set |
| conflict\_counter\_th | O | 1 | Indication to user and fsm\_core\_top that there were conflicts (from the same id) over the maximum and the accelerator need to return to idle and restart the proccess |
| IDs | O | 12x256 | The labling of the BBoxes |
| valid\_id | O | 1 | Indication to user that the ids outputs are valid to read |

Table 5: Oflow-Top - Pins Description

## 2.4. Clocks and Resets

There is one clock for all of the chip, the period of this clock is 5[ns].

The reset is negative, and it is called reset\_N.

## 2.5. Interfaces description

The accelerator has interface with 2 units:

* 1. CPU: the cpu is the main processing unit, which manages the reg-file unit of the accelerator. This unit accesses the reg-file via AMBA-APB protocol.

This unit will get back the labled BBoxes from the accelerator.

* 1. Glue-logic: the glue-logic is responseble for connecting the accelerator with the DMA that stores the BBoxes. This unit fetches from the DMA 24 BBoxes concurrently and transfers them to the accelerator.

It controls the new\_set and new\_frame.

The inputs from the accelerator are: ready\_new\_set, ready\_new\_frame, conflict\_counter\_th.

## 2.6. Sub-units description

### 2.6.1 oflow\_core

**Block diagram**

**A computer screen shot of a computer chip

Description automatically generated**

Figure 12 : Oflow-Core - block diagram

**Block description (+flow-chart)**

The oflow\_core module is the main unit of the accelerator, handling multiple tasks like feature extraction, object registration, and conflict resolve. It integrates several submodules (PEs), working in parallel to process data related to bounding boxes (bboxes) of objects across frames.

**Overview**

* **Purpose**: The oflow\_core module processes sets of bboxes received from a DMA and calculates scores for object similarity across frames. It assigns unique IDs to objects and handles cases where object identities may conflict.

**Features**

1. **Bounding Box Processing**: The module processes multiple bboxes simultaneously, leveraging hardware parallelism (using multiple Processing Elements, PEs). BBoxes are evaluated based on feature similarity (shape, color, position, etc.), and scores are computed.
2. **State Machines and Control**:
   * Several Finite State Machines (FSMs) coordinate different tasks:
     + *Core FSM*: Handles the overall workflow, including reading from memory and writing to it, feature extraction and registration.
3. **Parallel Processing with PEs**: The module uses multiple PEs (processing elements) to handle operations in parallel. Each PE works on a subset of BBoxes to accelerate processing, making it scalable for large datasets. Each PE consists of feature\_extraction and registration units. (The registration unit will be described later here, and the feature\_extraction was described in part A of this project).
4. **Conflict Resolve**: The module resolves conflicts that arise when multiple objects might share similar characteristics (e.g., overlapping bounding boxes). The conflict resolve logic ensures that each object gets a unique ID (until order 2), and alerts when there are at least 10 BBoxes with conflicts.
5. **Buffer Memory Management**: The module reads and writes bbox data to/from memory buffers. This is necessary because the system operates on multiple frames and tracks objects across time.

**Pins description**

|  |  |  |  |
| --- | --- | --- | --- |
| Signal\_Name | Type | Width | Description |
| set\_of\_bboxes\_from\_dma | I | [85:0]x256 | Input array of bounding boxes features from DMA. Each element corresponds to a bounding box feature. |
| new\_set\_from\_dma | I | 1 | Signal indicating that a new set of bounding boxes is ready from Glue\_Logic. |
| start | I | 1 | Start signal for indication to start the process. |
| new\_frame | I | 1 | Signal indicating that a new frame is ready from Glue\_Logic. |
| ready\_new\_set | O | 1 | Indicates the Glue\_Logic that the accelerator is ready to receive a new set of bounding boxes from DMA. |
| ready\_new\_frame | O | 1 | Indicates the Glue\_Logic that the accelerator is ready to receive a new frame from DMA. |
| conflict\_counter\_th | O | 1 | Signal indicating that the number of conflicts has been reached the threshold. |
| iou\_weight | I | 10 | This parameter will be set by the user by apb\_pwdata. It will determine the weight of iou\_metric. |
| w\_weight | I | 10 | This parameter will be set by the user by apb\_pwdata. It will determine the weight of width\_metric. |
| h\_weight | I | 10 | This parameter will be set by the user by apb\_pwdata. It will determine the weight of height\_metric. |
| color1\_weight | I | 10 | This parameter will be set by the user by apb\_pwdata. It will determine the weight of color1\_metric. |
| color2\_weight | I | 10 | This parameter will be set by the user by apb\_pwdata. It will determine the weight of color2\_metric. |
| dhistory\_weight | I | 10 | This parameter will be set by the user by apb\_pwdata. It will determine the weight of history\_metric. |
| score\_th\_for\_new\_bbox | I | 32 | This parameter will be set by the user by apb\_pwdata. It will determine the score threshould for defining BBox as a new BBox. |
| num\_of\_history\_frames | I | 3 | This parameter will be set by the user by apb\_pwdata. It will determine the number of history frames that will be compared with current frame. (1-5) |
| max\_threshold\_for\_conflicts | I | 4 | This parameter will be set by the user by apb\_pwdata. It will determine the number of conflicts from the same id until conflict\_counter\_th flag raises. |
| num\_of\_bbox\_in\_frame | I | 8 | Number of bounding boxes in the current frame. |
| valid\_id | O | 1 | Signal indicating that the output bounding box IDs are valid. |
| ids | O | [11:0]x256 | Array of output bounding box IDs, one for each detected object in the frame. |

Table 6: Oflow-core - Pins Description

**oflow\_pe**

**A screenshot of a computer

Description automatically generated**

Figure 13: oflow-pe - block diagram

**Block description (+flow-chart)**

This unit consists of 2 units: oflow\_feature\_extraction and oflow\_registration.

The oflow\_feature\_extraction was described in detail in part A.

The oflow\_registration will be described later in this part.

The main purpose of this unit is to extract the features of the BBoxes in the current frame and comparing them to the BBoxes in the history frames (from the mem\_buffer) to assist in the labeling process.

This block communicates with two main units with the help of interfaces.

One is for the connection with the oflow\_mem\_buffer\_wrapper (interface\_mem\_buffer\_pe) and the other is for connection with the oflow\_conflict\_resolve (interface\_cr\_pe).

For accelerate the calculation process there are 24 PEsthat work concurrently. Each PE serves one BBox in the current set of the current frame.

**Pins description**

|  |  |  |  |
| --- | --- | --- | --- |
| Signal\_Name | Type | Width | Description |
| iou\_weight | I | 10 | This parameter will be set by the user by apb\_pwdata. It will determine the weight of iou\_metric. |
| w\_weight | I | 10 | This parameter will be set by the user by apb\_pwdata. It will determine the weight of width\_metric. |
| h\_weight | I | 10 | This parameter will be set by the user by apb\_pwdata. It will determine the weight of height\_metric. |
| color1\_weight | I | 10 | This parameter will be set by the user by apb\_pwdata. It will determine the weight of color1\_metric. |
| color2\_weight | I | 10 | This parameter will be set by the user by apb\_pwdata. It will determine the weight of color2\_metric. |
| dhistory\_weight | I | 10 | This parameter will be set by the user by apb\_pwdata. It will determine the weight of history\_metric. |
| bboxes\_from\_dma |  | 86 | Input array of bounding boxes features from DMA. Each element corresponds to a bounding box feature. |
| num\_of\_pe | Input | 5 | The serial number of the current PE. |
| ready\_new\_frame | Input | 1 | Indicates the Glue\_Logic that the accelerator is ready to receive a new frame from DMA. |
| ready\_new\_set | Input | 1 | Indicates the Glue\_Logic that the accelerator is ready to receive a new set of bounding boxes from DMA. |
| flg\_for\_sampling\_last\_set | Input | 1 | Flag indicating if the last set should be sampled. (because there isn’t ready\_new\_set in the last set). |
| num\_of\_sets | Input | 4 | Total Number of data sets to process in each frame. |
| frame\_num | Input | 8 | The serial number of the current frame. |
| start\_fe | Input | 1 | Signal to start feature extraction. |
| not\_start\_fe | Input | 1 | Signal indicating to not start feature extraction in the specific PE (only in last set). |
| start\_registration | Input | 1 | Signal to start object registration. |
| not\_start\_registration | Input | 1 | Signal indicating to not start object registration in the specific PE (only in last set). |
| done\_pe | Input | 1 | Indicates the completion of calculations for all the sets in the frame. |
| done\_read\_to\_pe | Input | 1 | Indicates that the read operation in the mem\_buffer is done. (in each set). |
| data\_to\_pe\_0 | Input | 145 | Data input to similarity\_metric\_0 in the PE (includes history field). We get signal for read mem\_buffer. |
| data\_to\_pe\_1 | Input | 145 | Data input to similarity\_metric\_1 in the PE (includes history field). We get signal for read mem\_buffer. |
| row\_sel\_to\_pe | Input | 4 | Row selection signal for PE, comes from oflow\_core\_fsm\_write. |
| row\_sel\_to\_pe\_from\_cr | Input | 4 | Row selection signal from conflict resolution unit. (for fill\_hist) |
| write\_to\_pointer\_to\_pe | Input | 1 | Write enable signal for pointer in score board that need to change to resolve conflict. |
| write\_to\_id\_to\_pe | Input | 1 | Write enable signal for ID in score board that need to change to set as a new BBox. |
| data\_from\_cr\_pointer\_to\_pe | Input | 1 | Data from conflict resolution unit for writing pointer in score board that need to change to resolve conflict. |
| data\_from\_cr\_id\_to\_pe | Input | 12 | Data from conflict resolution unit for writing ID in score board that need to change to set as a new BBox. |
| row\_to\_change\_to\_pe | Input | 4 | Row selection signal for changing score board data when there is a conflict / new id. |
| done\_fe | Output | 1 | Indicates the completion of feature extraction.  (stays on ‘1’ until new start\_fe). |
| done\_registration | Output | 1 | Indicates the completion of object registration. |
| done\_score\_calc | Output | 1 bit | Indicates the completion of score calculation. |
| control\_for\_read\_new\_line | Output | 1 bit | Control signal to oflow\_core\_fsm\_read to start reading a new line of data. This signal is asserted to '1' two cycles before done similarity metric to prepare new line of data to be ready for the next similarity metric calculation. |
| data\_out\_pe | Output | 142 | Data output from PE for writing to memory. (without d\_history) (concatenates 2 pes data to 1 row) |
| id\_out | Output | 12x11 | Output array containing IDs from score board. |
| score\_to\_cr\_from\_pe | Output | 32 | Score output from PE to conflict resolution unit. |
| id\_to\_cr\_from\_pe | Output | 12 | ID output from PE to conflict resolution unit. |

**Core FSMs**

The oflow\_core consists of 5 FSMs that orchestrate the integration between the rest of the units. Below is their list:

* oflow\_fsm\_core\_top
* oflow\_fsm\_core\_fe
* oflow\_core\_fsm\_registration
* oflow\_core\_fsm\_read
* oflow\_core\_fsm\_write

**oflow\_fsm\_core\_top**

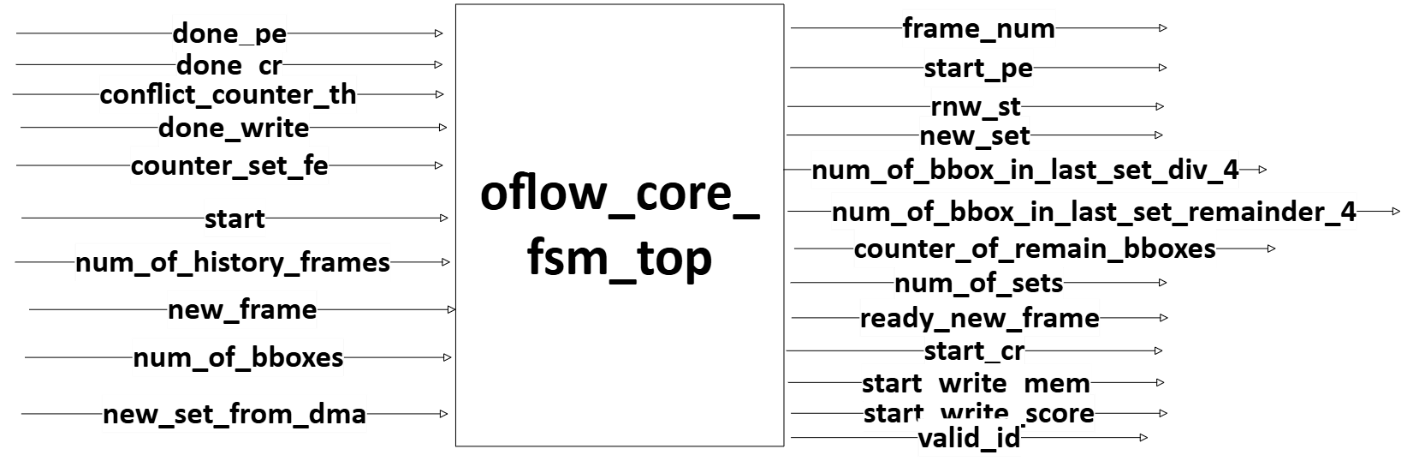


Figure 14: oflow\_core\_fsm\_top - block diagram

A diagram of a program

Description automatically generated

Figure 15: oflow\_core\_fsm\_top - FSM diagram

The oflow\_core\_fsm\_top module is responsible for controlling the overall flow of operations in a hardware system that performs optical flow, particularly focusing on feature extraction, registration, and conflict resolve. This module acts as the top-level finite state machine (FSM) for coordinating interactions between various submodules and managing data flow and memory operations. Here's a summary of its states:

* The FSM operates through multiple states, including idle, set\_variables, pe, conflict\_resolve, and write.
* In the idle\_st state, the system awaits new frame.
* In set\_variables\_st, it sets up the necessary variables for processing based on the number of bounding boxes in the frame.
* The pe\_st state activates processing elements (PEs) to handle the feature extraction and registration process.
* If there are conflicts in the data, the conflict\_resolve\_st state handles resolution.
* Finally, the write\_st state manages writing the processed results to memory.

**oflow\_fsm\_core\_fe**

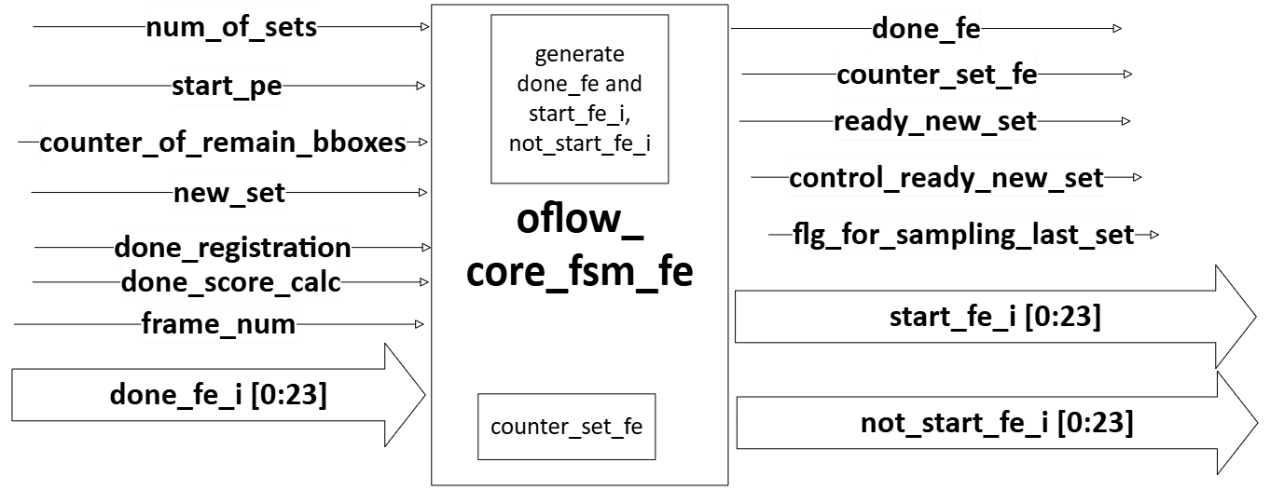


Figure 16: oflow\_core\_fsm\_fe - block diagram

The oflow\_core\_fsm\_fe module is responsible for managing the feature extraction (FE) phase of an optical flow system using a finite state machine (FSM).

* The module distributes the workload across multiple PEs, activating a subset of them based on the number of bounding boxes remaining.
* Manages synchronization signals (done\_fe, ready\_new\_set) to ensure proper progression to the next stage.

**oflow\_fsm\_core\_registration**

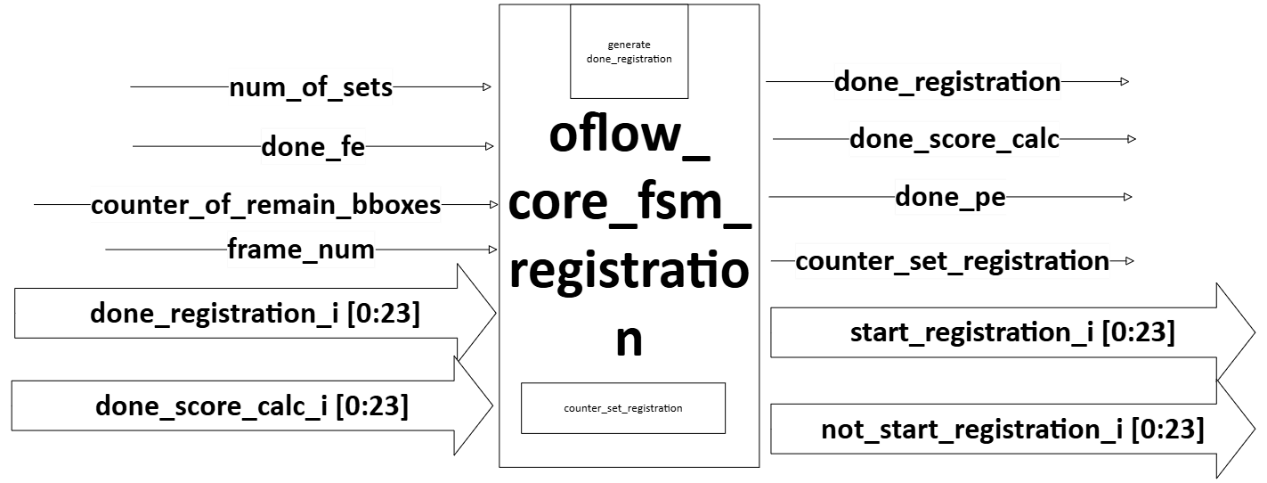


Figure 17: oflow\_core\_fsm\_registration - block diagram

The oflow\_core\_fsm\_registration module handles the registration and score calculation processes.

* Tracks the number of sets processed with a counter (counter\_set\_registration).
* Manages PE activation based on the number of bounding boxes remaining.
* Ensures that registration and score calculation are completed for all sets before progressing.

Overall, the module efficiently controls the registration and score calculation phases using a structured FSM to manage multiple sets and coordinate completion across PEs.

**oflow\_fsm\_core\_read**

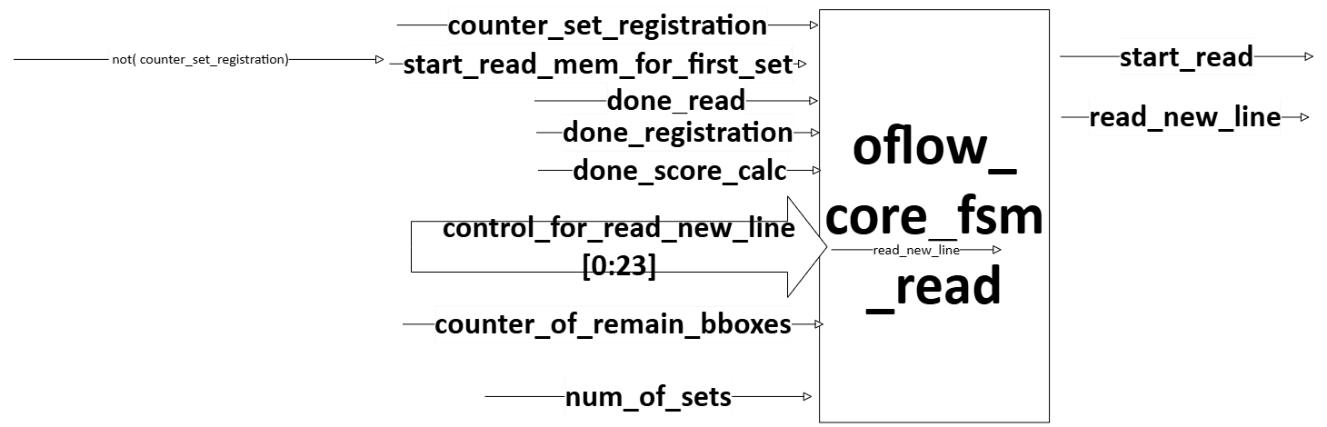
****

Figure 18: oflow\_core\_fsm\_read - block diagram

The oflow\_core\_fsm\_read module is a finite state machine (FSM) responsible for managing the memory read operations in an optical flow system. Its main roles include controlling the start and progression of memory reads, determining when to read new data lines, and coordinating with other FSMs involved in registration.

Functionality:

* The module coordinates memory reads based on the registration.
* It manages transitions between reading data lines and waiting for registration to complete.
* The state machine tracks the number of sets processed using a counter (counter\_set\_registration) and ensures the process continues until all sets are read.

Overall, this module efficiently controls memory read operations, synchronizing data access with other core processes in the system.

**oflow\_fsm\_core\_write**

**A white and black background with black text

Description automatically generated**

Figure 19: oflow\_core\_fsm\_write - block diagram

The oflow\_core\_fsm\_write module is a finite state machine (FSM) that manages the writing process in an optical flow system. It coordinates the selection of rows (each row represents different set) and processing elements (PEs) for writing data to the memory.

* The FSM processes bounding boxes in groups of 4 (because 2 PEs are written together to one row and the writing is to two rows – dual port sram), iterating through rows and PEs.
* It manages transitions between PEs, ensuring each PE streams out the appropriate portion of the data.
* It also handles cases where the last set of bounding boxes does not perfectly divide by 4, adjusting the state machine to process the remaining BBoxes.
* The module outputs control signals (ready\_from\_core, row\_sel, pe\_sel, and remainder) to coordinate writing with the memory buffer.

In summary, this module orchestrates the writing of bounding box data (features + id) into memory, ensuring proper distribution across rows and PEs while handling cases where the number of bounding boxes doesn't perfectly align with the PEs.

### 2.6.2 Registration

**Block diagram**

Figure 20: oflow\_registration - block diagram

**Block description (+flow-chart)**

The oflow\_registration module is a core component of the optical flow system that manages the registration (labeling) process by handling data flow, score calculation, and interaction between feature extraction, memory buffers, and scoreboards. This module is responsible for coordinating various elements such as feature extraction data, score calculation logic, and memory updates to track object registration across frames.

**Submodule Interactions:**

The oflow\_registration module instantiates and coordinates several submodules, each playing a distinct role:

1. **oflow\_score\_calc:** This submodule calculates similarity scores between current and previously feature extraction data. It compares these scores to find the minimum scores and outputs the results (min\_score\_0, min\_id\_0, etc.).
2. **oflow\_score\_board:** This submodule keeps track of the best-matching scores and their corresponding object IDs. It also communicates with the CR to update object registration information and buffer IDs.
3. **oflow\_registration\_fsm:** This FSM orchestrates the overall registration process, managing the start and stop conditions for score calculation, scoreboarding, and feature extraction.
4. **oflow\_registration\_feature\_extraction\_reg\_sb:** Handles feature extraction data by storing the current BBox features and registering the extracted data to feed it into the memory for subsequent calculations.

**Summary of Operation:**

The oflow\_registration.sv module coordinates the optical flow registration process by interacting with various modules, each responsible for score calculation, feature extraction, and memory management. It synchronizes feature extraction data with the score calculation logic, updates the score board, and outputs the final registration results for each BBox in the current frame. The FSM plays a key role in orchestrating this process, ensuring that registration happen smoothly across multiple frames and data sets.

**Pins description**

|  |  |  |  |
| --- | --- | --- | --- |
| Signal\_Name | Type | Width | Description |
| cm\_concate\_cur | I | 22 | Concatenated feature data of the current frame.  (TBD in oflow\_feature\_extraction) |
| position\_concate\_cur | I | 44 | Concatenated position (X\_TL, Y\_TL, X\_BR, Y\_BR) of the bounding box in the current frame. |
| width\_cur | I | 8 | Width of the bounding box in the current frame. |
| height\_cur | I | 8 | Height of the bounding box in the current frame. |
| color1\_cur | I | 24 | RGB color data of the bounding box (object) in the current frame. |
| color2\_cur | I | 24 | RGB color data of the bounding box (background) in the current frame. |
| done\_read | I | 1 | Signal indicating the end of the read operation from mem\_buffer (reading all the history frames from the mem\_buffer) |
| data\_to\_similarity\_metric\_0 | I | 145 | Concatenated feature data of the previous frame, including cm, position, width, height, color and history data. (left part of the row in the mem\_buffer – 1 BBox) |
| data\_to\_similarity\_metric\_1 | I | 145 | Concatenated feature data of the previous frame, including cm, position, width, height, color and history data. (right part of the row in the mem\_buffer – 1 BBox) |
| control\_for\_read\_new\_line | O | 1 | Control signal to oflow\_core\_fsm\_read to start reading a new line of data. This signal is asserted to '1' two cycles before done similarity metric to prepare new line of data to be ready for the next similarity metric calculation. |
| iou\_weight | I | 10 | Weight for the IoU metric in the final similarity score calculation. |
| w\_weight | I | 10 | Weight for the width difference metric in the final similarity score calculation. |
| h\_weight | I | 10 | Weight for the height difference metric in the final similarity score calculation. |
| color1\_weight | I | 10 | Weight for the first color channel difference metric in the final similarity score calculation. |
| color2\_weight | I | 10 | Weight for the second color channel difference metric in the final similarity score calculation. |
| dhistory\_weight | I | 10 | Weight for the d\_history\_metric in the final similarity score calculation. |
| id\_out | O | 12x11 | Output array of IDs from the score board |
| ready\_new\_frame | I | 1 | Indicates the Glue\_Logic that the accelerator is ready to receive a new frame from DMA. |
| row\_sel\_to\_pe | I | 4 | Row selection signal for PE, comes from oflow\_core\_fsm\_write. |
| frame\_num | I | 8 | The serial number of the current frame. |
| ready\_new\_set | I | 1 | Indicates the Glue\_Logic that the accelerator is ready to receive a new set of bounding boxes from DMA. |
| flg\_for\_sampling\_last\_set | I | 1 | Flag indicating if the last set should be sampled. (because there isn’t ready\_new\_set in the last set). |
| num\_of\_sets | I | 4 | Total Number of data sets to process in each frame. |
| done\_fe | I | 1 | Indicates the completion of feature extraction.  (stays on ‘1’ until new start\_fe). |
| start\_registration | I | 1 | Signal to start registration process |
| not\_start\_registration | I | 1 | Signal indicating to not start object registration in the specific PE (only in last set). |
| done\_registration | O | 1 | Indicates the completion of object registration. |
| done\_score\_calc | O | 1 | Indicates the completion of score calculation. |
| num\_of\_pe | I | 5 | The serial number of the current PE. |
| data\_out\_pe | O | 142 | Data output from PE for writing to memory. (without d\_history) (concatenates 2 pes data to 1 row) |
| row\_sel\_to\_pe\_from\_cr | I | 4 | Row selection signal from conflict resolution unit. (for fill\_hist) |
| write\_to\_pointer\_to\_pe | I | 1 | Write enable signal for pointer in score board that need to change to resolve conflict. |
| data\_from\_cr\_pointer\_to\_pe | I | 1 | Data from conflict resolution unit for writing pointer in score board that need to change to resolve conflict. |
| row\_to\_change\_to\_pe | I | 4 | Row selection signal for changing score board data when there is a conflict / new id. |
| write\_to\_id\_to\_pe | I | 1 | Write enable signal for ID in score board that need to change to set as a new BBox. |
| data\_from\_cr\_id\_to\_pe | I | 12 | Data from conflict resolution unit for writing ID in score board that need to change to set as a new BBox. |
| score\_to\_cr\_from\_pe | O | 32 | Score output from PE to conflict resolution unit. |
| id\_to\_cr\_from\_pe | O | 12 | ID output from PE to conflict resolution unit. |

Table 7: oflow\_registration - pins description

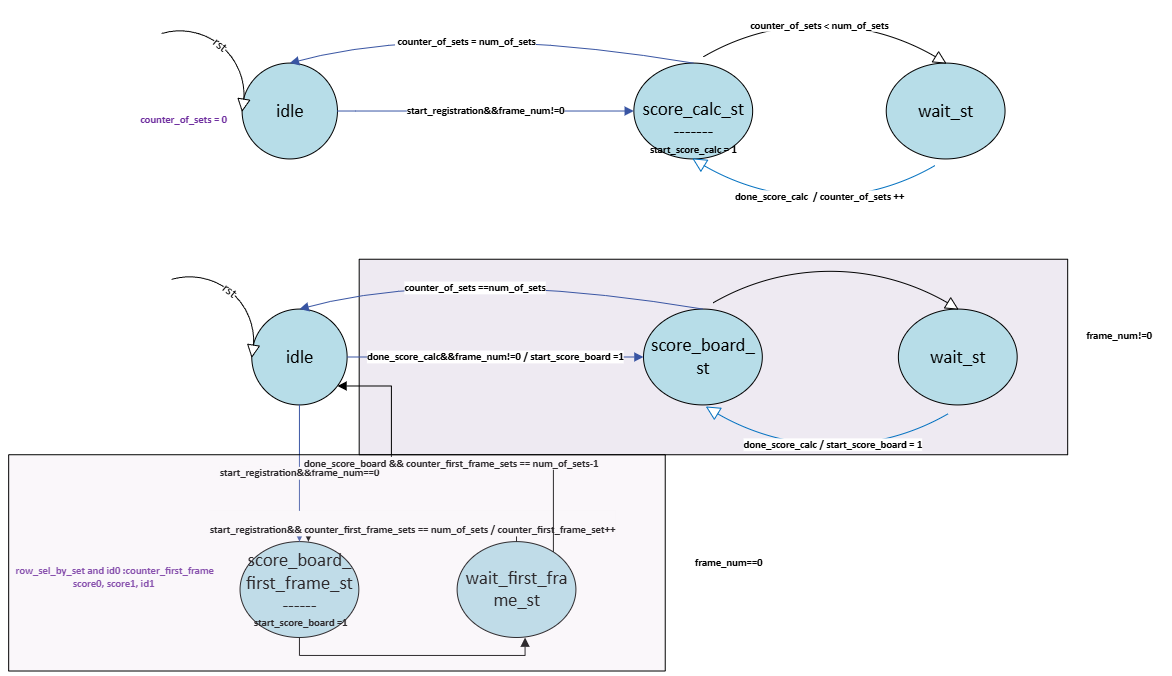
**FSM**

Figure 21: oflow\_registration - FSM

The oflow\_registration\_fsm is responsible for orchestrating the registration process. This FSM interacts with score calculation, scoreboarding, and in a multi-set environment, particularly during the registration of BBoxes across frames.

**Internal Logic:**

* **Counters (counter\_of\_sets, counter\_first\_frame\_sets):** These counters track the sets being processed during registration. Depending on whether it's the first frame or not, either counter\_first\_frame\_sets or counter\_of\_sets is used to control the row selection for the scoreboard.
  + row\_sel\_by\_set: Chooses the row based on the current frame. For subsequent frames (frame\_num != 0), counter\_of\_sets is used; otherwise, counter\_first\_frame\_sets is employed. This signal is for writing to the score\_board in the appropriate register.
  + id\_first\_frame: Computes the IDs for the first frame BBoxes using the number of PEs and the current set.

**Instantiated Modules:**

1. **oflow\_registration\_fsm\_score\_board:**

The oflow\_registration\_fsm\_score\_board is a subcomponent of the overall registration FSM and it specifically manages interactions with the scoreboard during the registration process. The module handles the transition between states, controls when the scoreboard updates, and manages the sets of frames, especially focusing on registration during both the first frame and subsequent frames.

1. **oflow\_registration\_fsm\_score\_calc:**

The oflow\_registration\_fsm\_score\_calc is a submodule within the larger oflow\_registration\_fsm design. Its role is to manage the FSM that controls the score calculation process during the registration phase. This module determines when the score calculation should start and incrementally tracks the sets being processed.

### 2.6.3 Conflict\_Resolve

**Block diagram**

A screenshot of a computer screen

Description automatically generated

Figure 22: oflow\_conflict\_resolve - block diagram

**Block description (+flow-chart)**

**Module Overview:**

The oflow\_conflict\_resolve module is designed to handle conflict resolution in an object registration hardware accelerator. Below is a description of the main features and functionalities:

The oflow\_conflict\_resolve module plays a critical role in managing conflicts that arise during the processing of BBoxes data. It interfaces with various components, such as a scoreboard and a LUT, to resolve conflicts between multiple BBoxes and ensure correct data handling.

**Features:**

1. **Lookup Table (LUT) Management:**
   * A built-in LUT is used to store temporary data related to conflict resolve. The module reads and writes to the LUT using signals such as data\_out\_lut\_for\_fsm, address\_lut, data\_in\_lut, and we\_lut.
   * The LUT addresses represent the ids of the BBoxes and the values in it represent the addresses in the histogram that count the instances of this ids in the current frame. For example: all the instances of id = 12 will be counted in the histogram at address ‘x’, therefore in the LUT at address 12 the value is ‘x’.
2. **Scoreboard Interface:**
   * The module interacts with the scoreboard, receiving scores (score\_to\_cr) and IDs (id\_to\_cr) to determine which bounding box data is involved in conflicts.
   * It provides control signals (row\_sel\_from\_cr, pe\_sel\_from\_cr, row\_to\_change, pe\_to\_change, etc.) to modify the scoreboard entries, ensuring that conflicting bounding boxes are handled correctly.
   * The module writes data back to the scoreboard, managing both pointer updates (write\_to\_pointer, in case of conflict) and ID updates (write\_to\_id, in case of new BBox) as part of the conflict resolution process.
3. **New Bounding Box (BBox) Processing:**
   * The module receives threshold values for new bounding boxes (score\_th\_for\_new\_bbox) and tracks counters (initial\_counter\_for\_new\_bbox) and total bounding boxes (total\_bboxes\_first\_frame) from the first frame of data.
   * These inputs ensure that new boxes are properly integrated into the processing pipeline.
4. **Flag Registers:**
   * The module employs flag registers to keep track of state information during conflict resolution. The flag register is reset when conflict resolve begins and is updated based on LUT writes (we\_lut).
5. **Conflict\_counter\_th:**
   * When the unit more than threshold value of conflicts, it arises a flag to indicate that the number of conflicts is over the threshold and the oflow\_core\_fsm\_top will jump to the idle state and the process is reset (frame\_num 🡪 0).

**Instantiations:**

1. **oflow\_conflict\_resolve\_fsm:**
   * A state machine module handles the core conflict resolution process, controlling the interactions with the scoreboard, LUT, and flag registers.
   * It operates based on the provided input signals, resolving conflicts and communicating the results via the scoreboard interface.
2. **Dual-Port RAM:**
   * The dual-port RAM is instantiated to provide LUT functionality, enabling the module to store and retrieve conflict resolution data.
   * The RAM is read from and written to as needed during conflict resolution, with proper control signals (CEB, WEB, CSB, OEB) managing access to the memory.

**Applications:**

This module is essential in scenarios where multiple BBoxes have the same labeling during real-time object registration processes. The conflict resolve mechanism ensures that data consistency is maintained until order two.

**Pins description**

|  |  |  |  |
| --- | --- | --- | --- |
| Signal\_Name | Type | Width | Description |
| start\_cr | I | 1 | Start signal for the conflict resolve process. |
| done\_cr | O | 1 | Indicates the completion of the conflict resolve process. |
| score\_th\_for\_new\_bbox | I | 32 | Score threshold for setting BBox as a new one (from the reg\_file). |
| initial\_counter\_for\_new\_bbox | I | 1 | Indicates that Initialization of the counter for the new bounding box is needed. (only in frame\_num = 0) |
| max\_threshold\_for\_conflicts | I | 4 | maximum number of conflicts from the same id until conflict\_counter\_th flag raises. (from the reg\_file). |
| total\_bboxes\_first\_frame | I | 8 | Total number of bounding boxes in the first frame. |
| score\_to\_cr | I | 32 | Score output from PE to conflict resolution unit. |
| id\_to\_cr | I | 12 | ID output from PE to conflict resolution unit. |
| row\_sel\_from\_cr | O | 4 | Row selection signal from conflict resolution unit. (for fill\_hist) |
| pe\_sel\_from\_cr | O | 5 | PE selection signal from conflict resolve unit (for fill\_hist) for reading from score board and insert the data into cr. |
| row\_to\_change | O | 4 | Row selection signal for changing score board data when there is a conflict / new id. |
| pe\_to\_change | O | 5 | PE selection signal for changing score board data when there is a conflict / new id. |
| data\_to\_score\_board\_ from\_cr\_pointer | O | 1 | Data from conflict resolution unit for writing pointer in score board that need to change to resolve conflict. |
| write\_to\_pointer | O | 1 | Write enable signal for pointer in score board that need to change to resolve conflict. |
| data\_to\_score\_board\_ from\_cr\_id | O | 12 | Data from conflict resolution unit for writing ID in score board that need to change to set as a new BBox. |
| write\_to\_id | O | 1 | Write enable signal for ID in score board that need to change to set as a new BBox. |
| conflict\_counter\_th | O | 1 | Signal indicating that the number of conflicts has been reached the threshold. |

**FSM**

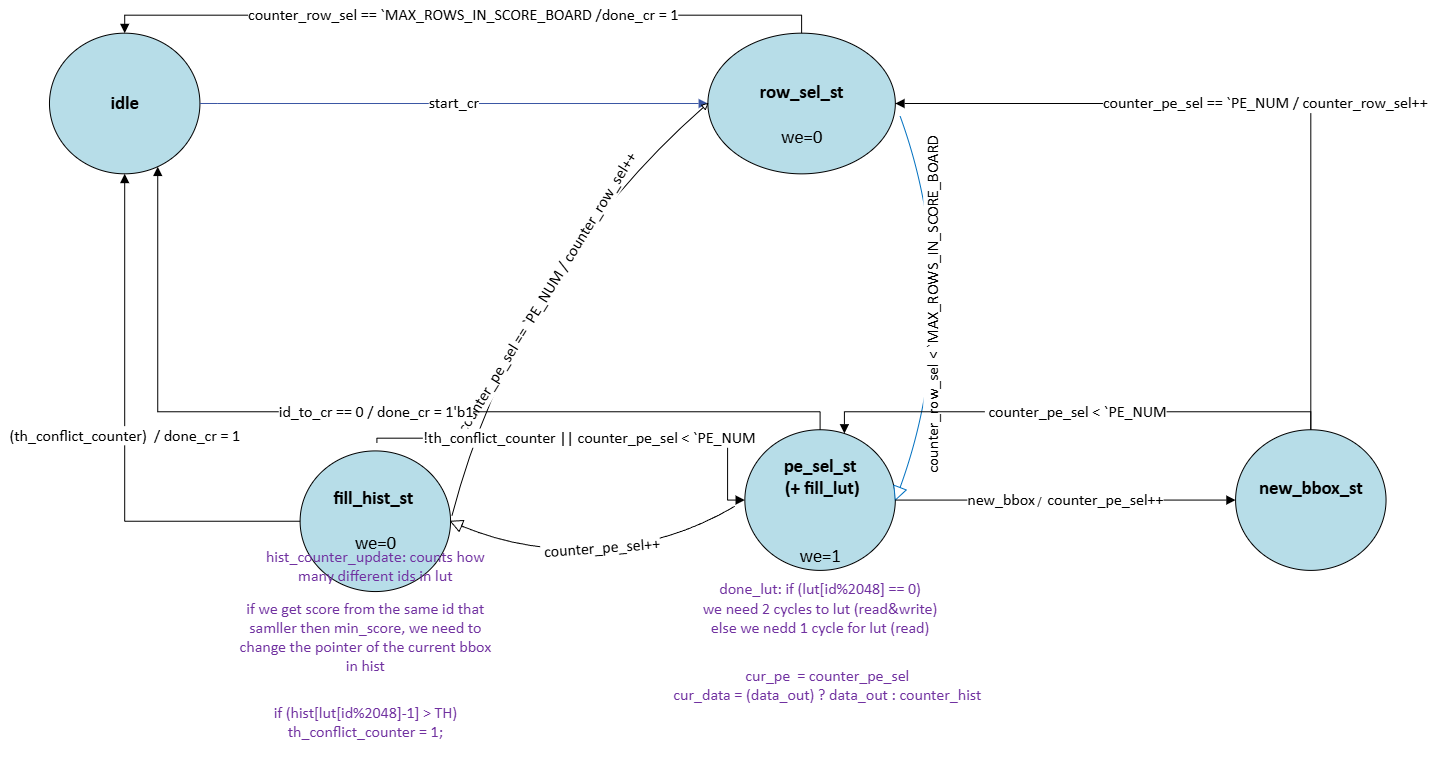


Figure 23: oflow\_conflict\_resolve - FSM

oflow\_conflict\_resolve\_fsm is responsible for handling conflicts that arise when attempting to resolve the object registration between multiple bounding boxes. The module likely operates within a larger framework for object registration, where it interacts with scoreboards, and flags to handle incoming data related to BBoxes in frames. The FSM communicates with the scoreboard to track information such as the scores, IDs, and selections of the PEs and rows involved in the conflict resolve.

**Features of the FSM:**

1. **FSM:** The module operates with several states:
   * **idle\_st:** The module waits for the conflict resolve to start.
   * **row\_sel\_st:** Selects a row to read from the scoreboard.
   * **pe\_sel\_N\_fill\_lut\_st:** Selects a PE and fills the LUT with relevant data.
   * **fill\_hist\_st:** Updates the history registers based on the current state of the LUT and data and determines if conflicts exist.
   * **new\_bbox\_st:** Handles the insertion of new BBoxes.

The FSM checks if the score associated with the bounding box is above a threshold (score\_th\_for\_new\_bbox) and updating the relevant ids in the score\_board. If conflicts from the same id exceed the threshold (max\_threshold\_for\_conflicts signal), the conflict\_counter\_th is flagged, the system moves back to the idle state and the accelerator starts from zero.

### 2.6.4 oflow\_mem\_buffer\_wrapper

A screenshot of a computer

Description automatically generated**Block diagram**

Figure 24: oflow-Mem-Buffer-Wrapper - block diagram

**Block description (+flow-chart)**

The oflow\_mem\_buffer\_wrapper module is a wrapper designed to manage memory operations, integrating both memory buffer and FSM logic to control it. Below is a detailed description of its main components and functionality:

Functions:

1. **Memory Management**: This module interfaces with a memory buffer to store and retrieve data across multiple frames.
2. **Control Signals**:
   * The wrapper uses control signals (rnw\_st, start\_read, start\_write) to handle read and write operations based on the current processing state.
   * The control signals are coordinated with a FSM that synchronizes the memory read and write processes (done\_read, done\_write).
3. **Processing Element (PE) Interface**:
   * Write state: Data inputs from four processing elements (data\_in\_0, data\_in\_1) are accepted and processed.
   * Read state: Corresponding outputs (data\_out\_0, data\_out\_1) are extracted after interacting with the memory buffer.

Internal Logic:

* **Write Enable (WE) Control**:  
  The module determines when to write to memory using we signal, which is set based on the rnw\_st (read/write) signal and the readiness of the core FSM (ready\_from\_core).
* **Chip Select and Output Enable**:  
  Two chip select signals (csb\_0, csb\_1) control the selection of memory banks for reading or writing, with an output enable signal (oeb) controlling when data is driven onto the output bus.
* **Offset Calculations**:  
  The module handles separate offsets (offset\_0, offset\_1) for read and write operations, ensuring proper addressing during memory access.

Module Instantiations:

1. **Memory Buffer (oflow\_MEM\_buffer)**:  
   The module instantiates an oflow\_MEM\_buffer that manages actual data storage, interfacing with the memory using various control signals and data inputs/outputs.
2. **FSM for Memory Buffer Control (oflow\_fsm\_mem\_buffer\_in\_mem\_wrapper)**:  
   An FSM controls the memory operations within the wrapper, synchronizing read and write cycles, and generating the necessary offsets for memory access.

Usage:

This module is intended to be used in designs where efficient memory management for multi-frame object registration is required. It provides an interface for reading and writing data to memory, manages the memory addressing and timing.

This modular design ensures flexibility in handling various object registration tasks while maintaining synchronization with external control units.

This unit also maintains start and end pointers to read only the lines in use from the memory.

**Example of Read & Write Transaction**

1. **Initialization:**
   * Assume frame\_num = 0, num\_of\_history\_frames = 3, and the buffer is reset to zero.

The MEM will be divided into sections: pointers [0] = 0, pointers [1] = 42, pointers [2] = 84.

1. **Data Input:**
   * Input data values for writing (for demonstration purposes):
     + data\_in\_0 = 284'b10101010… (example input for port 0)
     + data\_in\_1 = 284'b11001100… (example input for port 1)
   * Address offsets: (offsets are updated by oflow\_fsm\_write\_buffer)
     + offset\_0 = 3
     + offset\_1 = 4
   * Write Enable:
     + we = 1 (indicating to perform write operation)
2. **Writing data:**
   * On the next clock edge, data will be written to the calculated addresses in the memory.
3. **Reading data:**
   * After the write operation, assume we signal is set to 0 (indicating to perform read operation).
   * The frame\_num gets frame\_num\_to\_read from oflow\_fsm\_buffer\_read and allows calculating which pointers to use based on the modulo operation with num\_of\_history\_frames.
   * For example, to perform a read operation after writing, based on offsets:
     + Calculated addresses would be:
       - addr\_0 = pointers [frame\_num % num\_of\_history\_frames] + offset\_0
       - addr\_1 = pointers [frame\_num % num\_of\_history\_frames] + offset\_1
   * The data can now be read from data\_out\_0 and data\_out\_1. (\*Only data\_out\_0 is used).
4. **Output:**
   * The output will return the values stored in the specified memory positions. After the write operation, the signals will be:
     + data\_out\_0 = 284'b10101010…
     + data\_out\_1 = 284'b11001100…

This example illustrates how the oflow\_MEM\_buffer module processes read and write operations based on the frame structure and input signals.

**Pins Description**

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal\_Name** | **Type** | **Width** | **Description** |
| read\_new\_line | I | 1 | Indicates when a new line can be read after the similarity metric 2 cycles before finishes processing one line |
| start\_read | I | 1 | Signal to start reading data |
| start\_write | I | 1 | Signal to start writing data |
| data\_in\_0 | I | 284 | Data input from processing element (PE), will be written to port 0 of the mem  \_buffer |
| data\_in\_1 | I | 284 | Data input from processing element (PE), will be written to port 1 of the mem  \_buffer |
| frame\_num | I | 8 | Serial number of the current frame (0-255) |
| num\_of\_history\_frames | I | 3 | Number of history frames to be stored as fallback |
| num\_of\_bbox\_in\_frame | I | 8 | Indicates the end of the frame memory |
| rnw\_st | I | 1 | Read/Write control signal (1 = read, 0 = write) |
| ready\_from\_core | I | 1 | Ready signal from the core FSM indicating it is ready for new writing |
| done\_read | O | 1 | Indicates the completion of a read operation (reading all the rows in the relevant history frames) |
| done\_write | O | 1 | Indicates the completion of a write operation (writing all the BBoxes in the current frame) |
| data\_out\_0 | O | 284 | Data output to the processing element (PE), port 0 (for reading) |
| data\_out\_1 | O | 284 | Data output to the processing element (PE), port 1 (Not in use because each pe contains 2 similarity\_metric units and each row in memory contains 2 BBoxes, therefore it is enough to use only one port) |
| counter\_of\_history\_frame\_to\_interface | O | 3 | Counter indicating the current history frame being sent to the interface to calc its d\_history |

Table 8: Oflow\_Mem\_Buffer\_wrapper - Pins Description

**FSM**

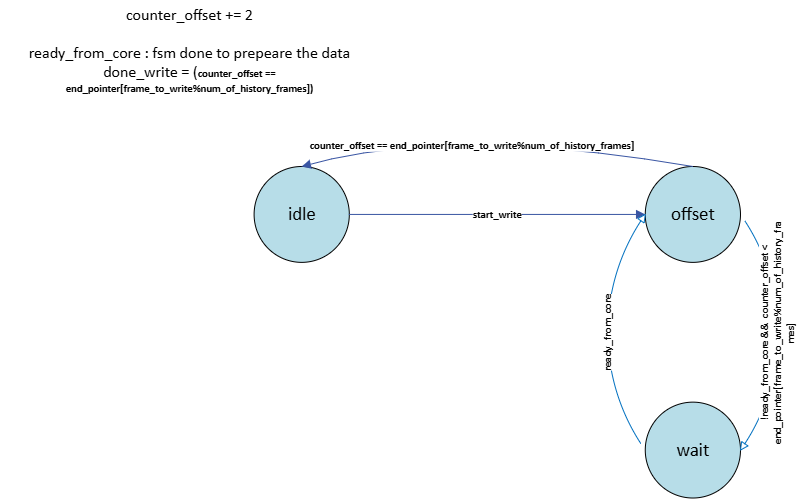
There are 2 FSMs to control the reading and writing to memory.

Each FSM contains 3 states.

The offset state is for generating the offsets to the mem\_buffer for creating the addresses.

In the write FSM, there is wait state to wait for ready\_from\_core.

In the read FSM, there is frame state to determine which history frame should be read.



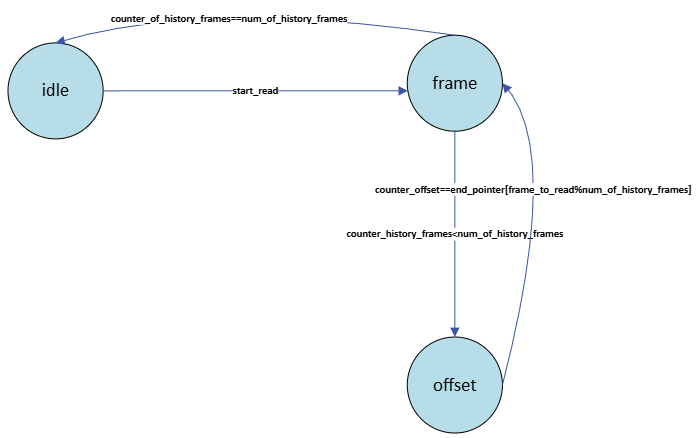


Figure 25: oflow\_mem\_buffer\_wrapper – FSM write

Figure 26: oflow\_mem\_buffer\_wrapper – FSM read

**Constraints of oflow\_MEM\_buffer\_wrapper**

* 1. There is a tradeoff between num\_of\_history\_frames and num of BBoxes in frame. The maximum product between them needs to be 256.

If the user chooses:

1)num of history frames = 1: num\_of\_BBoxes = 256

2)num of history frames = 2: num\_of\_BBoxes = 128

3)num of history frames = 3: num\_of\_BBoxes = 84

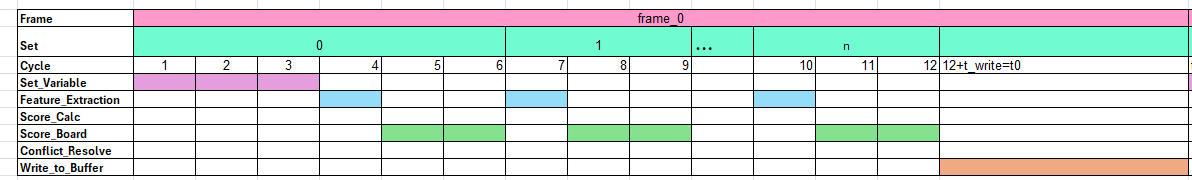
4)num of history frames = 4: num\_of\_BBoxes = 64

5)num of history frames = 5: num\_of\_BBoxes = 50

## 2.7. Performance

It can be observed from the performance tables that the unit with the greatest impact is the pe. This unit consists of feature\_extraction, score\_calc and score\_board.

This series of operations takes the most time – t\_sc.

****

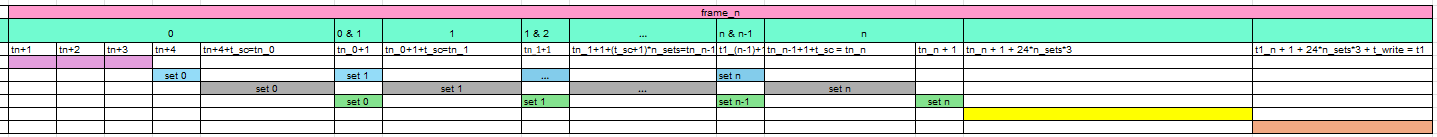
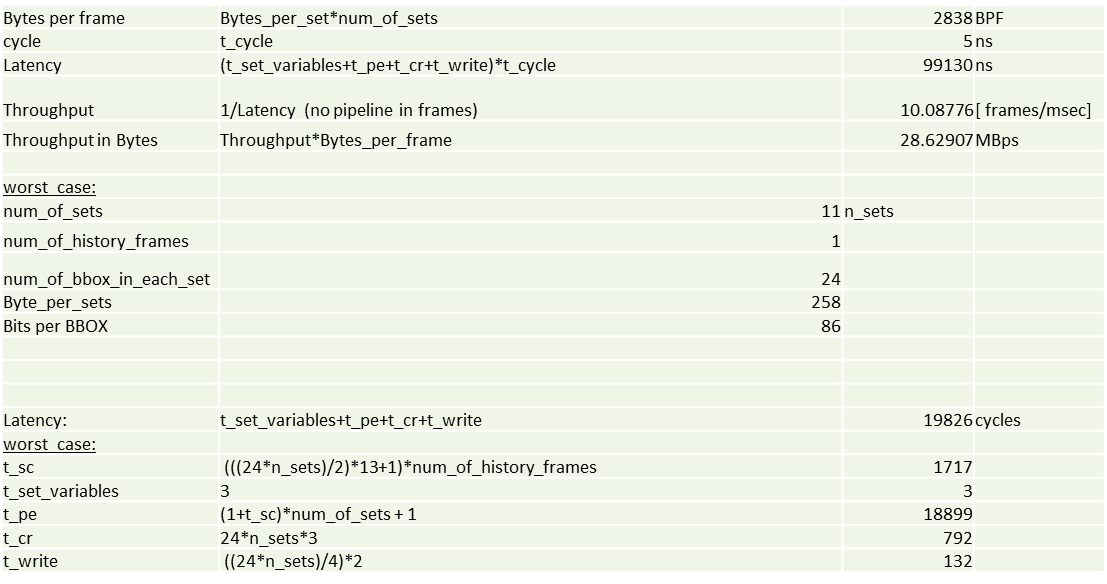
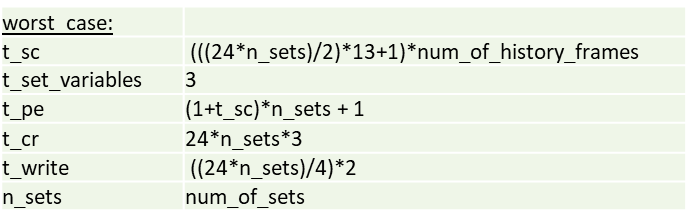
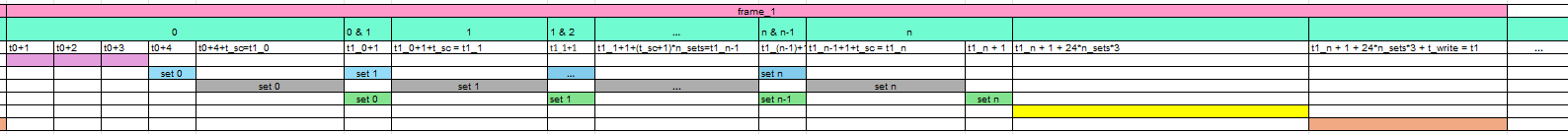
****

Figure 27: Performance



The throughput obtained is approximately 10 frames per millisecond, which is about 29 megabytes per second.

💡For comparison, ByteTrack algorithm processes 30 frames per second (FPS) on a single NVIDIA V100 GPU.

Calculation:

1. Bytes per frame:  
   256 objects/frame \* 10 bytes/object = 2,560 bytes/frame
2. Frames per second: 30 fps
3. Throughput in bytes per second:  
   2,560 bytes/frame \* 30 frames/second = 76,800 bytes/second 77 kBps

References:

<https://blog.roboflow.com/what-is-bytetrack-computer-vision/>

<https://www.ecva.net/papers/eccv_2022/papers_ECCV/papers/136820001.pdf>

<https://ar5iv.labs.arxiv.org/html/2110.06864>

<https://www.datature.io/blog/introduction-to-bytetrack-multi-object-tracking-by-associating-every-detection-box>

https://www.luffca.com/2023/06/multiple-object-tracking-bytetrack/

<https://blog.qure.ai/notes/byte-track-multi-object-tracking>

<https://www.tredence.com/blog/advancements-in-object-tracking-bytetrack-vs-fairmot>

https://www.ikomia.ai/blog/bytetrack-multi-object-tracking

## 2.8. Synthesis

# 2.8.1 Technology and Constrains (SDC)

# 2.8.2 Floorplan

2.8.3 Area and Static Power

2.8.4 Clock Tree Synthesis (CTS)

## A book with text on it Description automatically generated 2.9. Programmer’s Guide

The initial step is to update the Reg-File unit.

The user needs to insert to this unit by the APB protocol the parameters:

* + - * Weights of the similarity metrics
      * Threshold value for the score that above it the BBox will be defined as a new BBox
      * Num of history frames: The user can choose how many frames to store in the history.

(There is a tradeoff between this number and the maximum number of BBoxes per frame)

* + - * Max\_threshold\_for\_conflicts: The user can choose the threshold value, and if the number of conflicts with the same id reaches this value, the accelerator resets.

# 3. Zero-order ("aliveness") verification

## 3.1. Test-Plan and Test-Results oflow\_top

|  |  |  |
| --- | --- | --- |
| **Test name** | **Description & Purpose** | **Inputs** |
| **1:** Multiple frames with conflicts under and over the threshold value and not full sets (Number of BBox in the last is less than 24) | Simulate several frames with multiple BBoxes, ensuring that the system maintains proper functionality like managing the conflicts.  The purpose is to test the accelerator’s ability to handle continuous registration, especially the conflict\_resolve over multiple frames with multiple BBoxes. | Frame 0: 72 BBoxes (3 full sets).  Frame 1: 70 BBoxes (3 sets) so there will be less than 10 conflicts in this frame.  Frame 2: 70 BBoxes (3 sets) – 6 BBoxes from each of the two first sets will be the most similar to a specific BBox from previous frame so there will be more than 10 conflicts.  Frame 3: 24 BBoxes (1 set) |
| 2: Multiple frames with new BBoxes to labeling and conflicts | Simulate several frames with multiple BBoxes, ensuring that the system maintains proper functionality like managing the new BBoxes.  The purpose is to test the accelerator’s ability to detect disappearing BBoxes from a frame after several frames and tagging it properly and detect new BBoxes and tagging them properly. | Frame 0: 72 BBoxes (3 full sets).  Frame 1: 70 BBoxes (3 sets) so 2 BBoxes (71,72) disappear.  Frame 2: 75 BBoxes (4 sets) – BBoxes (71,72) return and 3 BBoxes are new. |

|  |  |  |
| --- | --- | --- |
| **Test name** | **Expected Outputs** | **Pass Criteria** |
| **1:** Multiple frames with conflicts under and over the threshold value and not full sets (Number of BBox in the last is less than 24) | IDs: Frame 0 – the IDs are expected to be in ascending order (1 🡪 72)  Frame 1 – the IDs of BBoxes that are similar to BBoxes in frame 0 need to be with the same ID and also the BBoxes which have conflicts need to be replaced to the second ID.  conflict\_counter\_th: In frame 2 there are more than 10 conflicts and therefore this signal should be raised, and the next frame need to be reset to frame\_num=0 | Correct transition between the states of the fsm\_core\_top.  Frame 0: idle -> set\_variables -> pe\_st -> write\_st ->  Frame 1: idle -> set\_variables -> pe\_st -> cr\_st -> write\_st  Frame 2: idle -> set\_variables -> pe\_st -> cr\_st  Frame 3: idle -> set\_variables -> pe\_st -> write\_st -> |
| 2: Multiple frames with new BBoxes to labeling and conflicts (and BBox that gets out the frame and returns after 3 frames) | IDs: Frame 0 – the IDs are expected to be in ascending order (1 🡪 72)  Frame 1 – the IDs of BBoxes that are similar to BBoxes in frame 0 need to be with the same ID until 70.  Frame 2 – the IDs of BBoxes that are similar to BBoxes in frames 0 and 1 need to be with the same ID (including BBoxes 71,72 that disappeared). And ID=73,74,75 for the new BBoxes. | Correct transition between the states of the fsm\_core\_top.  Frame 0: idle -> set\_variables -> pe\_st -> write\_st ->  Frame 1: idle -> set\_variables -> pe\_st -> cr\_st -> write\_st  Frame 2: idle -> set\_variables -> pe\_st -> cr\_st -> write\_st  -detecting the return of BBoxes 71,72 and the new BBoxes in frame 2 |

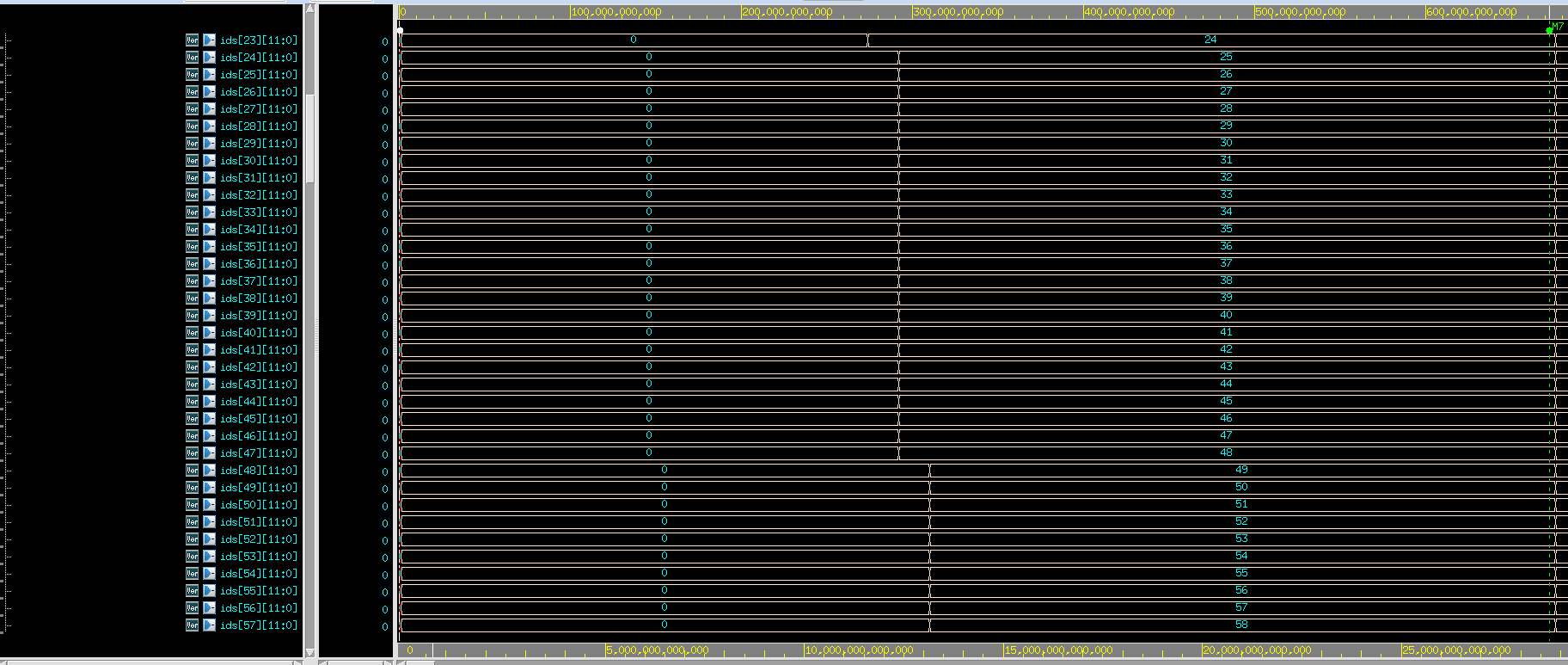
Table 10: oflow\_top - test plan

**Test 1:**

Frame\_0: As expected, the states are idle -> set\_variables -> pe\_st -> write\_st.

In addition, the ids in ascending order (1 🡪 72). (3 full sets).

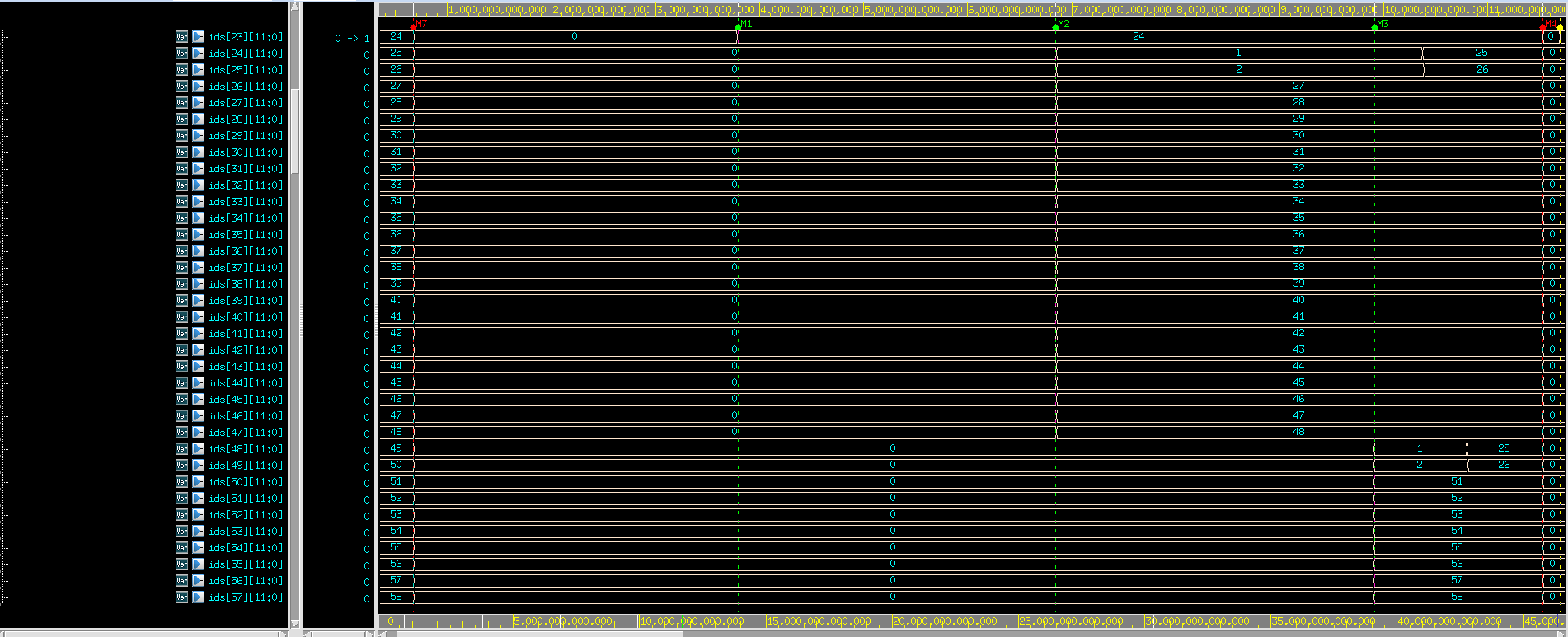
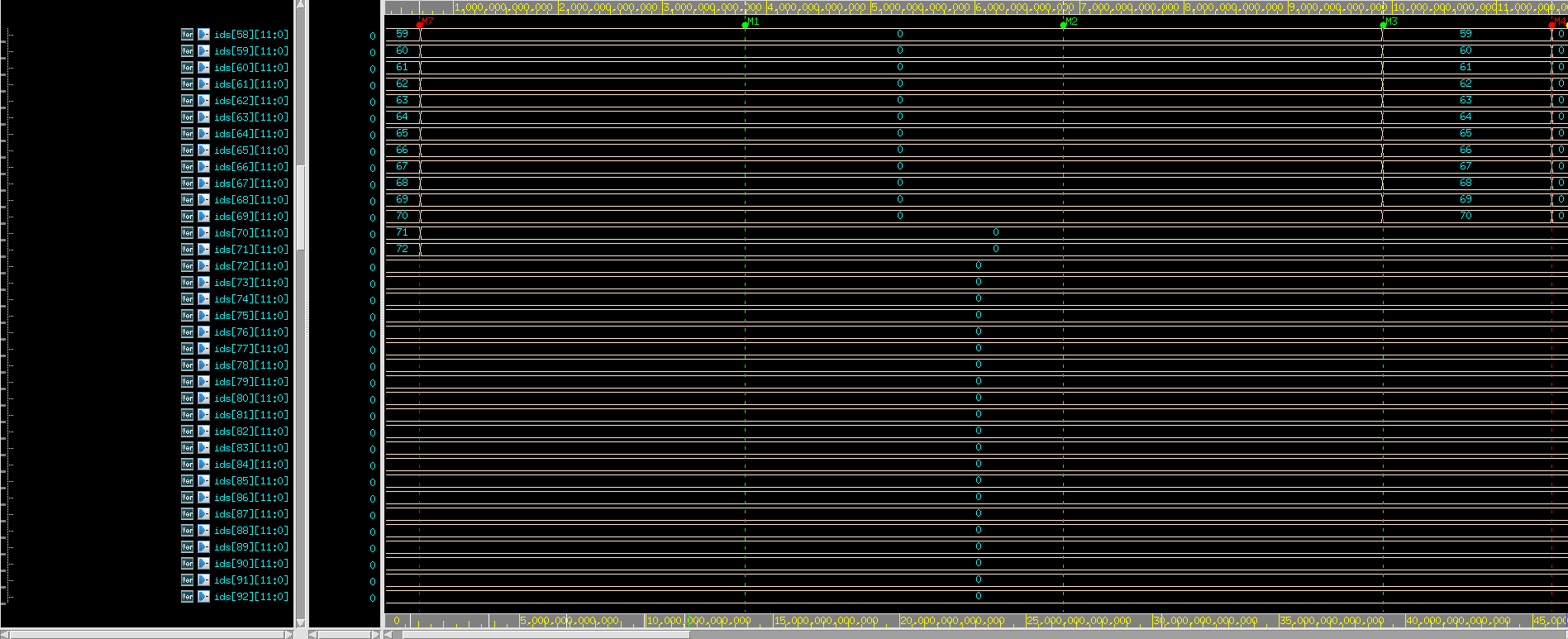
Figure 28: oflow\_top – test 1 results – frame 0



Frame\_1: As expected, the states are idle -> set\_variables -> pe\_st -> cr\_st -> write\_st.

In addition, there are 70 BBoxes in this frame that will be labeled according to the similar BBoxes in the previous frame.

Figure 29: oflow\_top – test 1 results - frame 1



Set 0: The labeling is the same as in set 0 of frame 0 because in this set the features are the most similar to the features of set 0 in frame 0.

Set 1: The labeling is the same as in set 1 of frame 0 because in this set the features are the most similar to the features of set 1 in frame 0. But in the conflict\_resolve state there are conflicts between the two first BBoxes in set 1 (25, 26) and the two first BBoxes in set 0 (1, 2). Therefore, the IDs of these two first BBoxes of the second set are changed to the IDs from order 2 which is 25, 26.

Set 2: The labeling is the same as in set 2 of frame 0 because in this set the features are the most similar to the features of set 2 in frame 0. But in the conflict\_resolve state there are conflicts between the two first BBoxes in set 2 (48, 49) and the two first BBoxes in set 0 (1, 2) and the two first BBoxes in set 1 (25, 26)- all of them got the same IDs accordingly (1, 2). Therefore, the IDs of these two first BBoxes of the third set are changed to the IDs from order 2 which is 25,26.

Frame\_2: As expected, the states are idle -> set\_variables -> pe\_st -> cr\_st.

In addition, there are 70 BBoxes in this frame that will be labeled according to the similiar BBoxes in the previous frame. In this frame, there are multiple BBoxes which are labeled with the same id. This will cause to more than 10 conflicts and then the flag of th\_conflict\_counter arises to ‘1’. Therefore, the current state is moving to idle\_st and the system is restarted. (frame\_num 🡪 0 again).

Figure 30: oflow\_top – test 1 results - frame 2

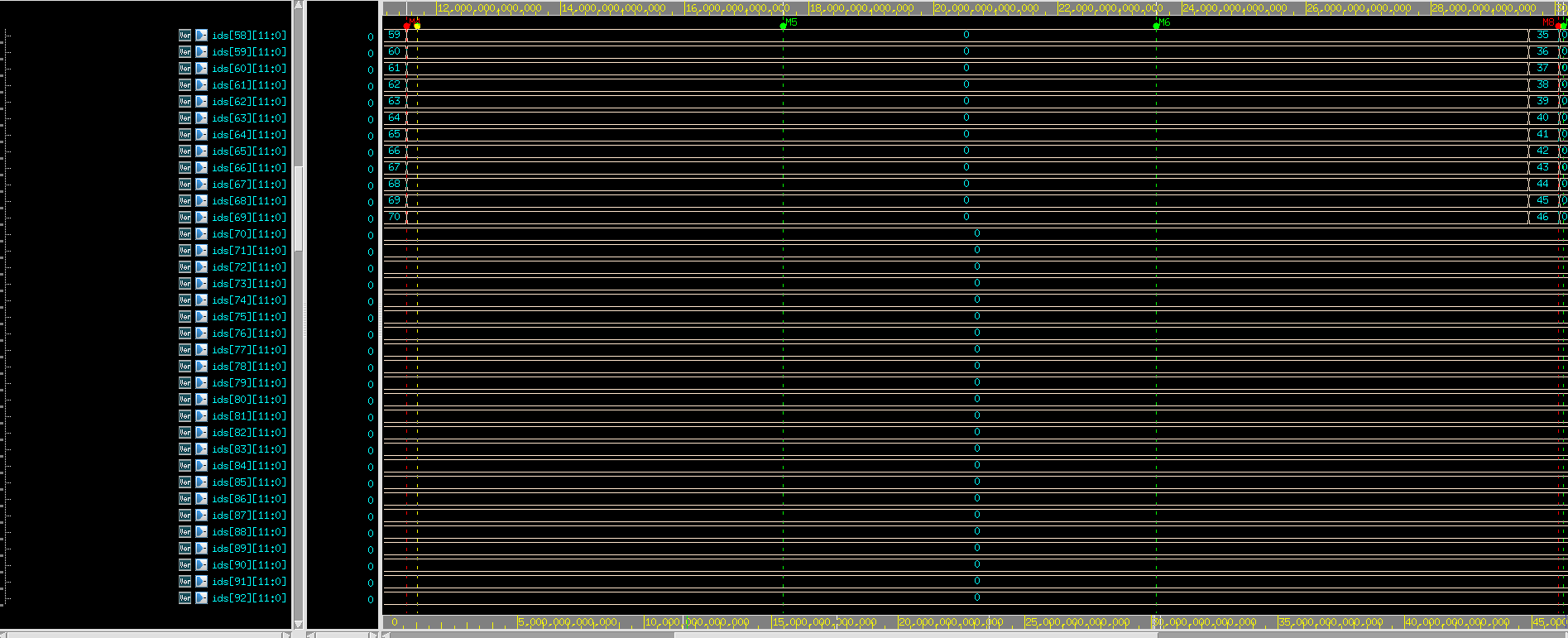
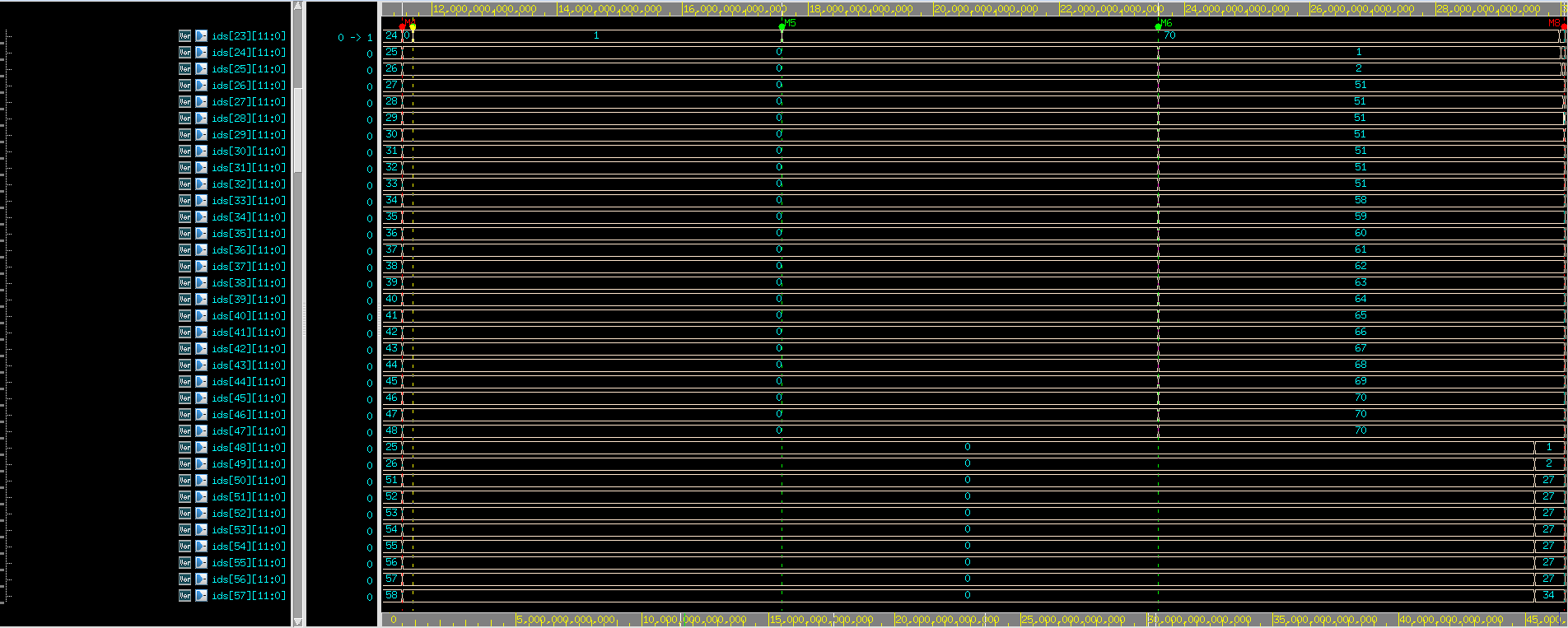
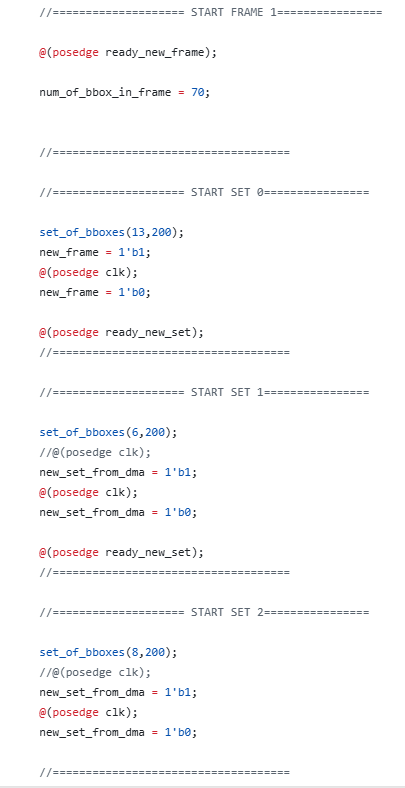


Figure 31: oflow\_top - test bench 1 - frame 1 and 2



In each set, the first two BBoxes are the same for all the frames.

The next BBoxes in each set have the same features expect of x coordinate that is changed between the sets and the frames.

Set 0: The features of this set are similar to the features in set 2 of frame 1. (49-70)

BBoxes 3-9: These BBoxes are similar to the third BBox of the third set in the previous frame (which was labeled as 51 in frame 1). Therefore, all of them get the same id (51 in the first priority and 52 in the second priority). This causes to 7 conflicts and in the conflict\_resolve state the ids except to the first one are changed to the second priority (52).

The rest of the BBoxes are labeled in ascending order, starting from 58 to 70.

.

Set 1: The features of this set are similar to the features in set 2 of frame 1.

BBoxes 27-33: These BBoxes are similar to the third BBox of the third set in the previous frame (which was labeled as 51 in frame 1). Therefore, all of them get the same id (51 in the first priority and 52 in the second priority). This causes to more 7 conflicts of the same ID (51) and in the conflict\_resolve state the ids except to the first one are changed to the second priority (52) until there are 10 conflicts from the same id (51) and then the flag of conflict\_counter\_th arises and the rest of the BBoxes with this conflict (30-33) don’t change to their second id. And the system is returned to idle and restart.

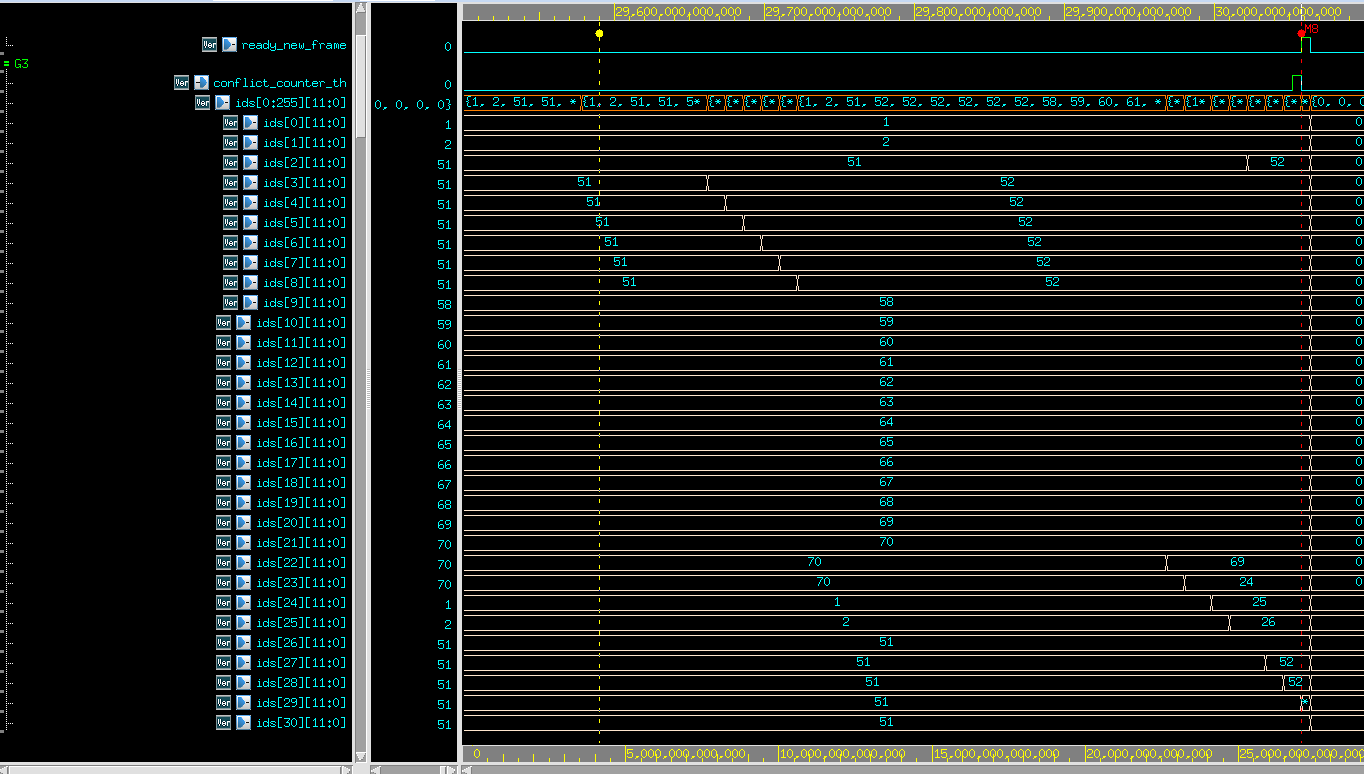


Figure 32: oflow\_top - test 1 results - frame 2 - set 1

Set 2: In this set there isn’t conflict\_resolve because the conflict\_resolve is done after handling the second set.

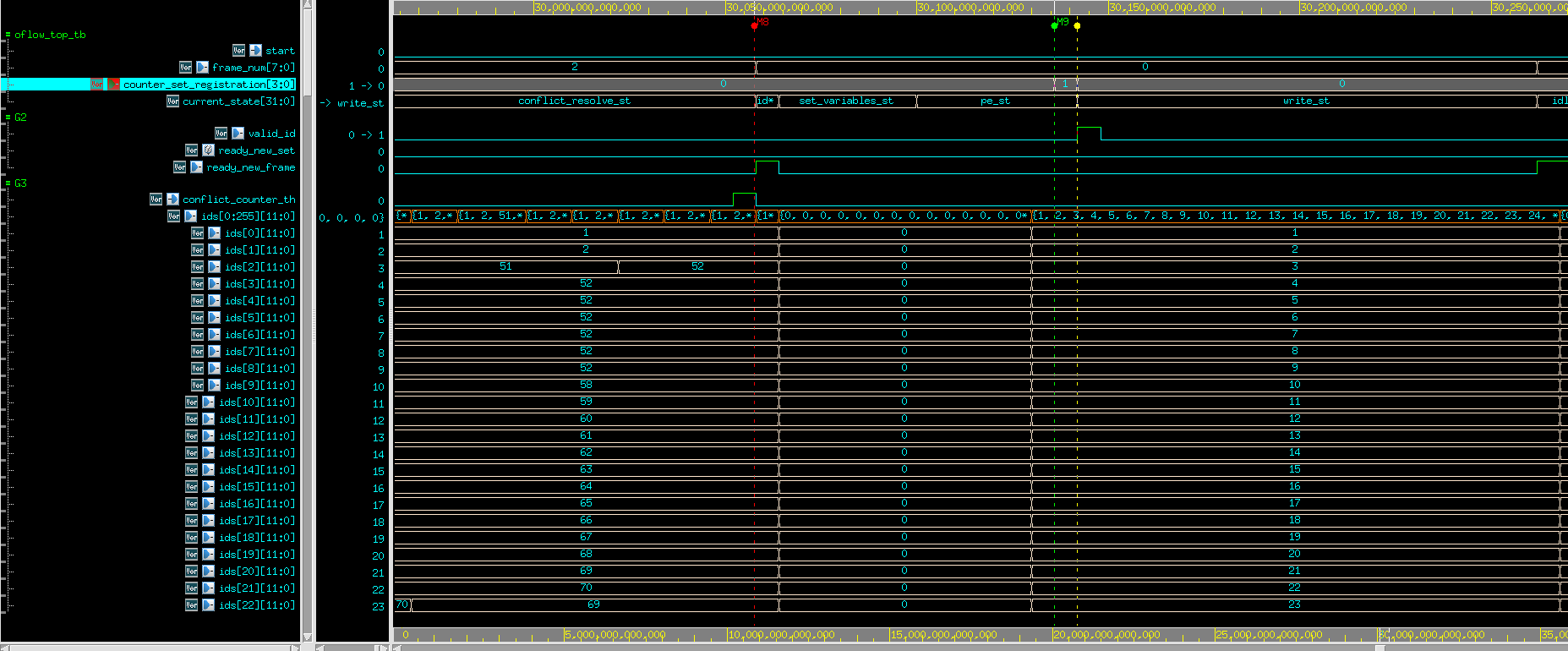
The features of this set are similar to the features in set 1 of frame 1. These BBoxes are supposed to be label as 1, 2, 27-46 (only 22 BBoxes in this set 🡪 until 46 instead of 48). But BBoxes 51-57 are labeled the same because all of them are similar to the same BBox (labeled as 27) and their id won’t change because the conflict\_resolve was done before.

Frame\_3: As expected, the states are idle -> set\_variables -> pe\_st -> write\_st.

In addition, the ids are in ascending order (1 🡪 24). (one full set).

The frame\_num is 0 again.

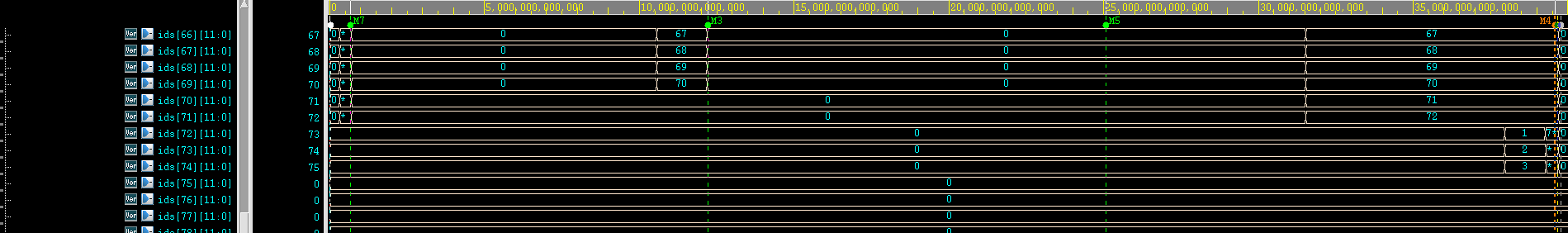
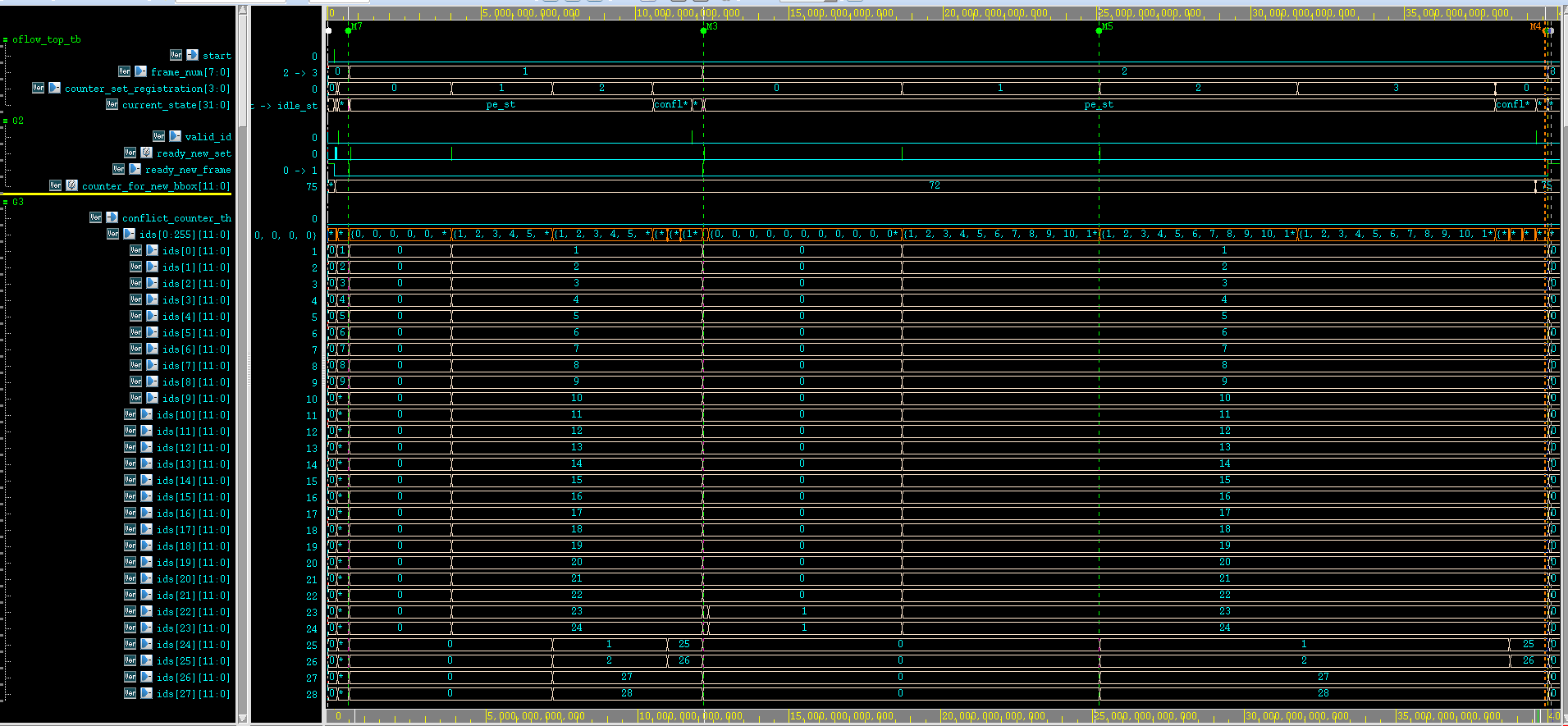
Figure 33: oflow\_top – test 1 results - frame 3



**Test 2:**

It can be seen here the outputs of oflow\_top for the sequence of frames in the second test:

Figure 34: oflow\_top – test 2 results



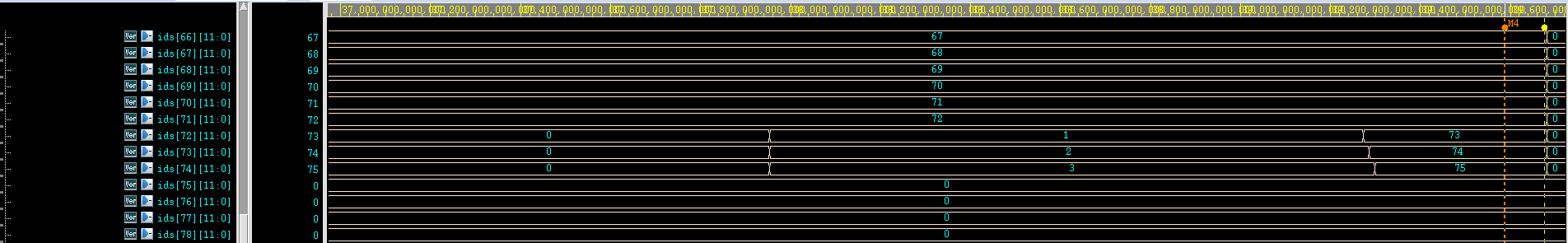


Figure 35: oflow\_top – test 2 results – frame 2 last set

As can be seen, BBoxes 71 and 72 were re-labeled after re-entering frame 2. Additionally, three new boxes were labeled as 73, 74, and 75 after the conflict\_resolve stage (where new BBoxes that achieved above a certain score threshold are labeled).

## 3.2. Test-Plan and Test-Results oflow\_registration

|  |  |  |
| --- | --- | --- |
| **Test name** | **Expected Outputs** | **Pass Criteria** |
| **1:** Multiple frames with conflicts under and over the threshold value and not full sets (Number of BBox in the last is less than 24) | In this test, the focus is on **frame 2, set 1, pe\_num=2**.  The signals:  start\_score\_calc is expected to rise ‘1’ after the previous done\_score\_board,  start\_score\_board of this current set is expected to rise ‘1’ the moment done\_score\_calc of the current set arises ‘1’.  Min\_id\_0,1 are expected to lock at the end of this set, on the IDs with the min scores.  The features of the current set are expected to be the same for all the process of the current set 1.  Data\_to\_similiraty\_metric\_0,1:  These are the features of the history frames, that arrive from the mem\_buffer and expected to change whenever the offset (address of the line, which is read) is changed.  Offset\_0:  Ther is only one relevant port for read state, because there are only 2 similiraty\_metric units, each for one BBox, and in each line in the mem\_buffer there are 2 BBox.  The offset is expected to rise in ascending order until ((#BBOX\_in\_previous\_frame/2)-1). Therefore, the offset is expected to be changed:  Frame\_1(70 BBoxes): 0🡪 34  Frame\_0(72 BBoxes): 0🡪35  The offset should be updated every time the control\_for\_read\_new\_line signal arises.  control\_for\_read\_new\_line signal arises: this signal should be arising each time it is needed to read new line from the mem\_buffer.  Done\_read: in the of reading all the relevant rows from the mem\_buffer, the mem\_buffer will return this signal to indicate that the processing of set 1 is finished. | In this test the focus will be on PE 2, set 1 in frame 2.  Min\_id\_0,1 need to be 51,52 accordingly with the minimum scores. |

Table 11: oflow\_registration - test plan

**Test 1:**

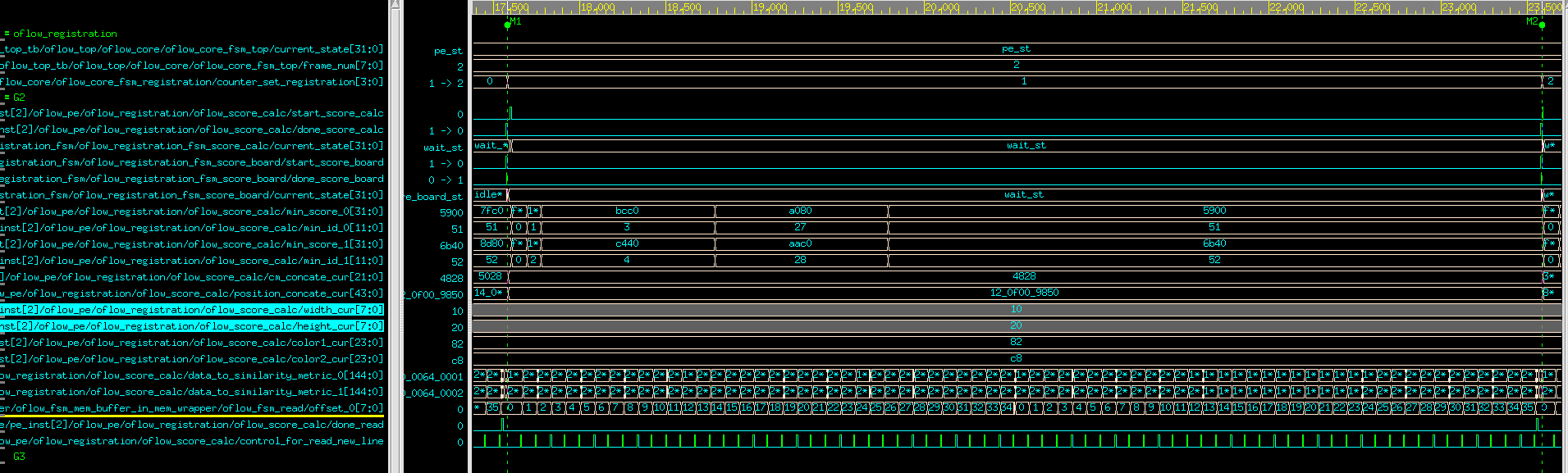


Figure 36: oflow\_registration – test 1 results (frame\_num 2, set 1, pe\_num 2)

As expected, min\_id\_0 is 51 and min\_id\_1 is 52. Why?

The third BBox in set 1 is inserted to PE 2. This BBox is the most similar to the third BBox of the third set in the previous frame (which was labeled as 51 in frame 1).

## 3.3. Test-Plan and Test-Results Conflict\_Resolve

|  |  |  |
| --- | --- | --- |
| **Test name** | **Expected Outputs** | **Pass Criteria** |
| **1:** Multiple frames with conflicts under and over the threshold value and not full sets (Number of BBox in the last is less than 24) | In this test, the focus is on frame 2.  The signals:  start\_cr is expected to rise ‘1’ after the pe\_st.  done\_cr is expected to rise ‘1’ after the conflict\_counter\_th arises too.  Id\_to\_cr: expected to change whenever the state is pe\_sel\_st. As shown in the waveform of oflow\_top – frame2, set1 there will be multiple BBoxes with the same id – 51.  score\_to\_cr: expected to change whenever the state is pe\_sel\_st,  row\_sel\_from\_cr, pe\_sel\_from\_cr:  row\_sel is supposed to raise from 0 to 2 but stops in 1 because of more than 10 conflicts.  Pe\_sel is supposed to raise from 0 to 24 in row\_sel=1 stops in 5 because of more than 10 conflicts that move the state to idle\_st.  Row\_to\_change and pe\_tp\_change: these signals need to store the row and pe of the BBox that “lose” in the conflict.  Data\_to\_score\_board\_from\_cr\_pointer and write\_to\_pointer:  Write\_to\_pointer is raised to ‘1’ when there is a conflict, and the other signal is raised to the index of the next priority which is 1.  Flag signals: data\_out\_flag is for indicate if the current id that has already been written to the LUT. address\_flag is the same as id\_to\_cr. data\_in\_flag is when the accelerator encountered the ids, in the current frame, for the first time. | In this test the focus is on frame 2.  current\_state of the fsm:  Idle 🡪 row\_sel -> (pe\_sel\_st 🡪 fill\_hist\_st) X 24 -> row\_sel 🡪 pe\_sel\_st X 6  coflict\_counter\_th arises to ‘1’ after 10 conflicts with the same id. Due to that the done\_cr should arise too.  Data\_out\_flag needs to be 2’b10 when there is a conflict on id = 51(<). |
| 2: Multiple frames with new BBoxes to labeling and conflicts (and BBox that gets out the frame and returns after 3 frames) | In this test, the focus is on frame 2.  The signals:  data\_to\_score\_board\_ from\_cr\_id:  need to change to the desired new id when there is new BBox  write\_to\_id:  need to raise every time there is new BBox  New\_bbox:  Flag that raises every time there is new BBox.  Counter\_for\_new\_bbox: In frame 0, this counter is reset to total\_bboxes\_in\_first\_frame, and increases by one after each new BBox | In this test the focus is on frame 2.  data\_to\_score\_board\_from\_cr\_id needs to be 73 -> 74 -> 75.  write\_to\_id and new\_bbox need to raise 3 times. (because there are 3 new BBoxes).  counter\_for\_new\_bbox: needs to be 72 after frame 0, and change in conflict\_resolve of frame 2 to 73 -> 74 -> 75. |

Table 12: oflow\_conflict\_resolve - test plan

**Test 1:**



Figure 37: oflow\_conflict\_resolve – test 1 results

It can be assumed that the same score\_to\_cr for multiple BBoxes is not as expected.

The reason behind this case is: in figure **\_\_\_\_\_\_\_\_\_\_\_\_\_\_** in the first set of frame 2, after the tenth BBox the features are the same to the features of the third set in frame 1 except the x coordinate that has the same difference for each of these BBoxes.

This can be shown on the next figure (figure **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**):

Figure 38: Inputs of the BBoxes in frame 1 and 2 – test 1

frame: 1

Set: 2

X: 8

frame: 2

Set: 0

X: 10



As expected, the pass critria for the conflict\_counter\_th and done\_cr is successfully passed.

In addition, the data\_out\_flag is changed as described in the pass criteria column.

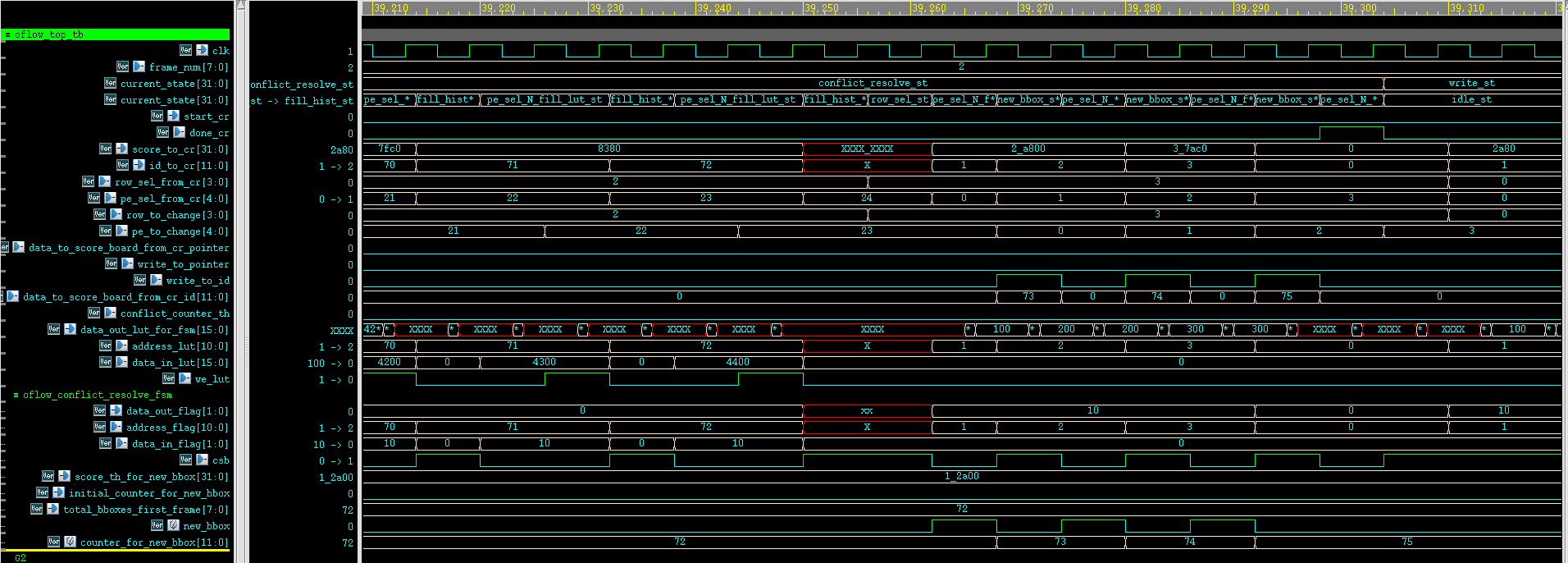
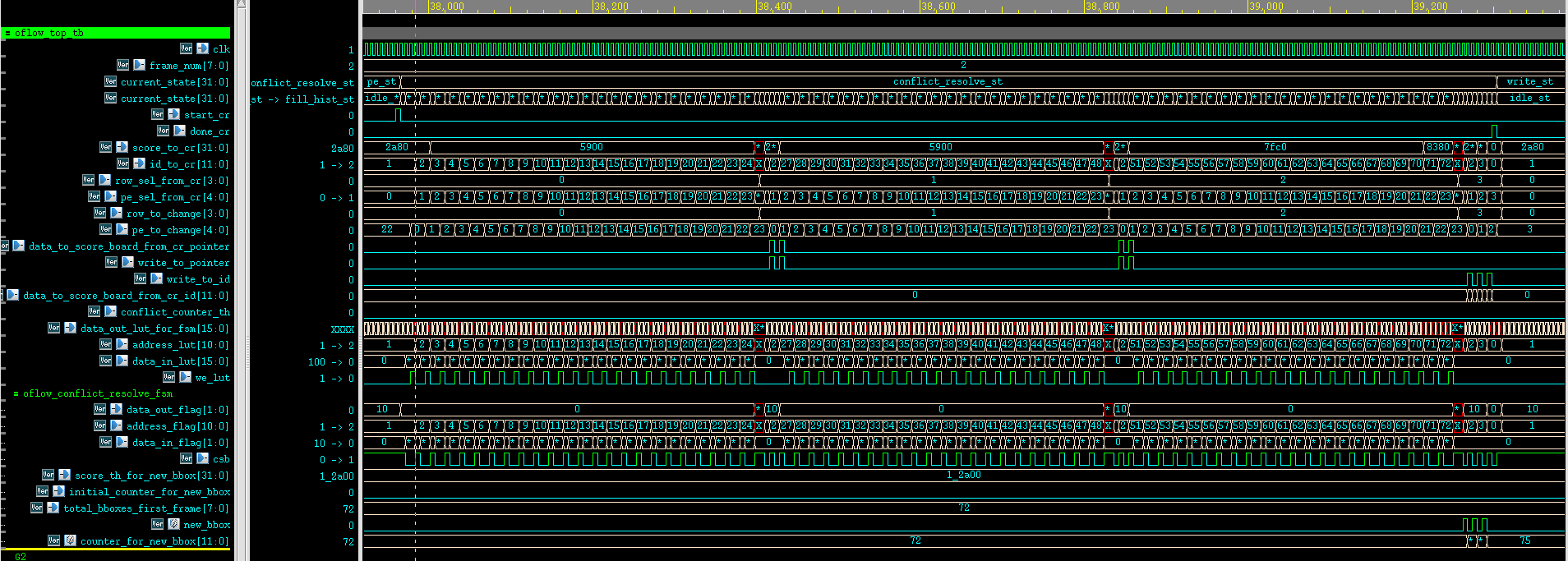
**Test 2:**

Figure 39: oflow\_conflict\_resolve – test 2 results – frame 2

Figure 40: oflow\_conflict\_resolve – test 2 results - frame 2 - zoom in

As can be seen, data\_to\_score\_board\_from\_cr\_id is 73 -> 74 -> 75 and write\_to\_id raises 3 times, 1 time for each new BBox (There is a need to change their id).

In addition, new\_bbox flag raises 3 times as it should.

Moreover, counter\_for\_new\_bbox is starting from 72 and when there is a new BBox, it increases by 1.

## 3.4. Test-Plan and Test-Results oflow\_mem\_buffer\_wrapper

|  |  |  |
| --- | --- | --- |
| **Test name** | **Expected Outputs** | **Pass Criteria** |
| **1:** Multiple frames with conflicts under and over the threshold value and not full sets (Number of BBox in the last is less than 24) | In this test, the focus is on frame 0 (72 BBoxes) and 1 (70 BBoxes).  The signals:  start\_write is expected to rise to ‘1’ after the pe\_st is done in frame 0 and after the cr\_st is done in frame 1.  done\_write expected to rise to ‘1’ after finishing writing all the BBoxes in the current frame.  Ready\_from\_core: expected to rise after the data (id from score\_board and features from the feature\_extraction\_reg\_sb) of the previous BBox was already written.  Data\_in\_0,1: expected to change each time to be ready for ready\_from\_core signal. (Each signal consists of 2 BBoxes, because there are 2 BBoxes in each row)  Current\_state of mem\_buffer\_fsm\_write: supposed to move from offset\_st (preparing the address) to wait\_st (writing the data) each time when there is a need for write new row)  Pointers: These pointers are initialized in frame\_num = 0. The main purpose of them is to pointer on the start row of each history frame’s section.  End\_pointers: The purpose of them is to pointer to the last row of each history frame’s section, which is supposed to be with written data.  Offset\_0,1: These signals run from 0 to end pointer of the current frame section in the mem\_buffer.  Addr\_0,1: These signals are the addresses of the rows in mem\_buffer that the data will be written to them.  Addr\_0,1= pointers[frame\_num%num\_of\_history\_frames]+offset\_0,1 | In this test the focus is on frame 0 and 1.  Frame 0:  -Ready\_for\_core should be raised 18 times because there are 72 BBoxes and each write is for 4 BBoxes.  -Pointers: num\_of\_history\_frames = 3, therefore pointers will be divided into 3 sections. There are 128 rows in the memory 🡪 pointers = 0, 42, 84.  -The addresses should meet the next formula: addr\_0,1 = 0 + offset\_0,1.  -offset0,1 need to reach the last row which is full data (according to end\_pointers).  Frame 1:  -Ready\_for\_core should be raised 18 times because there are 70 BBoxes and each write is for 4 BBoxes. (0 will be written to the last row).  -The addresses should meet the next formula: addr\_0,1 = 42 + offset\_0,1.  -offset0,1 need to reach the last row which is full data (according to end\_pointers). |

Table 13: Oflow\_mem\_buffer\_wrapper - test plan 1 - write

Frame 0:



Figure 41: oflow\_mem\_buffer\_wrapper – test 1 results - write - frame 0

As expected, there are 18 arises of the signal ready\_from\_core.

In addition, the pointers are as expected: 0, 42, 84, 0, 0. And end\_pointers[0] is set to 36, which represent 36\*2=72 BBoxes.

Addr\_0 = offset\_0 because this is frame 0. (as well for addr\_1).

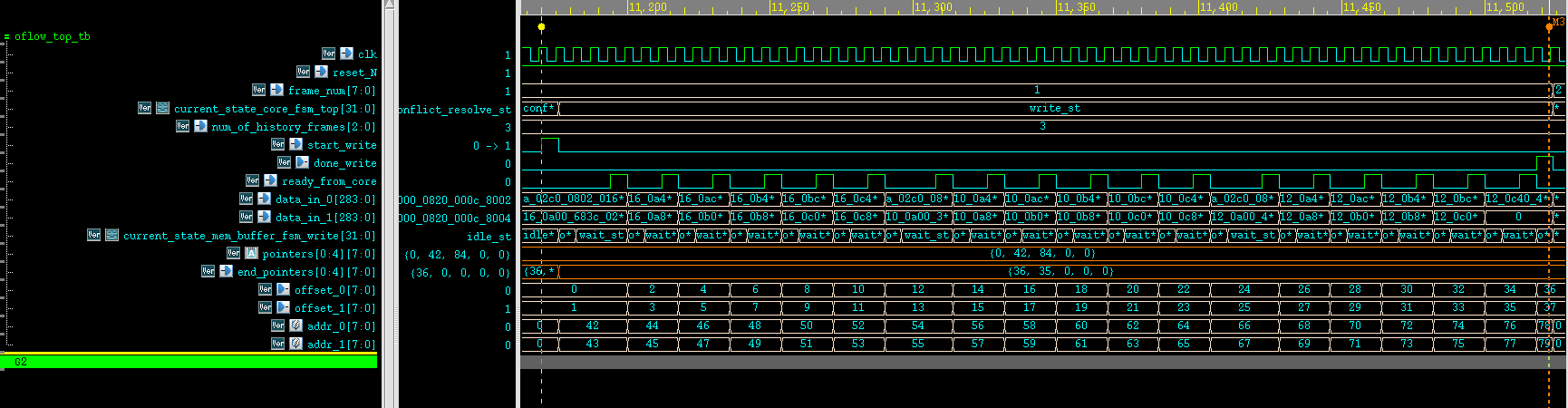
Frame 1:

Figure 42: oflow\_mem\_buffer\_wrapper - test 1 results - write - frame 1

As expected, there are 18 arises of the signal ready\_from\_core.

offset\_0 reaches to 34 (0-34, in increments of 2, is 35 rows which store 35\*2=70 BBoxes).

Offset reaches to 35 (1-35, in increments of 2). For offset\_1=35, which is not needed (because there are only 70 BBoxes), the data\_in\_1 that will be written to this row will be 0.

Addr\_0 = 42 + offset\_0 because this is frame 1. (as well for addr\_1).

|  |  |  |
| --- | --- | --- |
| **Test name** | **Expected Outputs** | **Pass Criteria** |
| **1:** Multiple frames with conflicts under and over the threshold value and not full sets (Number of BBox in the last is less than 24) | In this test, the focus is on frame 0 (72 BBoxes) and 2 (70 BBoxes).  The signals:  start\_read is expected to rise to ‘1’ when the first similarity\_metric of the set starts.  done\_read expected to rise to ‘1’ after finishing reading all the history frames from the mem\_buffer.  Similarity\_metric\_flag\_read\_new\_line: expected to rise 2 cycles before the similarity\_metric is done to prepare the next data from the mem\_buffer to the next process of similarity\_metric.  Data\_out\_0: one row from the mem\_buffer which contains 2 BBoxes (features+id).  Current\_state of mem\_buffer\_fsm\_read: supposed to move from offset\_st (reading all the rows of a specific history frame) to frame\_st (prepare the next history frame start\_pointer) each time when there is a need for read new frame)  Pointers: These pointers are initialized in frame\_num = 0. The main purpose of them is to pointer on the start row of each history frame’s section.  End\_pointers: The purpose of them is to pointer to the last row of each history frame’s section, which is supposed to be with written data.  Offset\_0: These signals run from 0 to the end\_pointer of the current frame section in the mem\_buffer.  Addr\_0: These signals are the addresses of the rows in mem\_buffer that the data will be written to them.  Addr\_0= pointers[frame\_num%num\_of\_history\_frames]+offset\_0 | In this test the focus is on frame 0 and 2.  Frame 0:  Verified that there is no read action because there aren’t history frames to compare with them.  Frame 2:  - Similarity\_metric\_flag\_read\_new\_line should be raised 71 times for each set of frame 2 because there are 70 BBoxes in frame 1 and 72 BBoxes in frame 0.  -The addresses should meet the next formula: addr\_0 = 42 + offset\_0 for reading frame 1 and addr\_0 = 0 + offset\_0 for reading frame 0  -offset 0 need to reach to 34 for reading frame 1 and 35 for reading frame 0.  - start\_read and done\_read are expected to rise to ‘1’ 3 times (1 for each set).  Frame\_to\_read is 1,0 because the reading process starts from the last frame. |

Table 14: Oflow\_mem\_buffer\_wrapper - test plan 1 - read

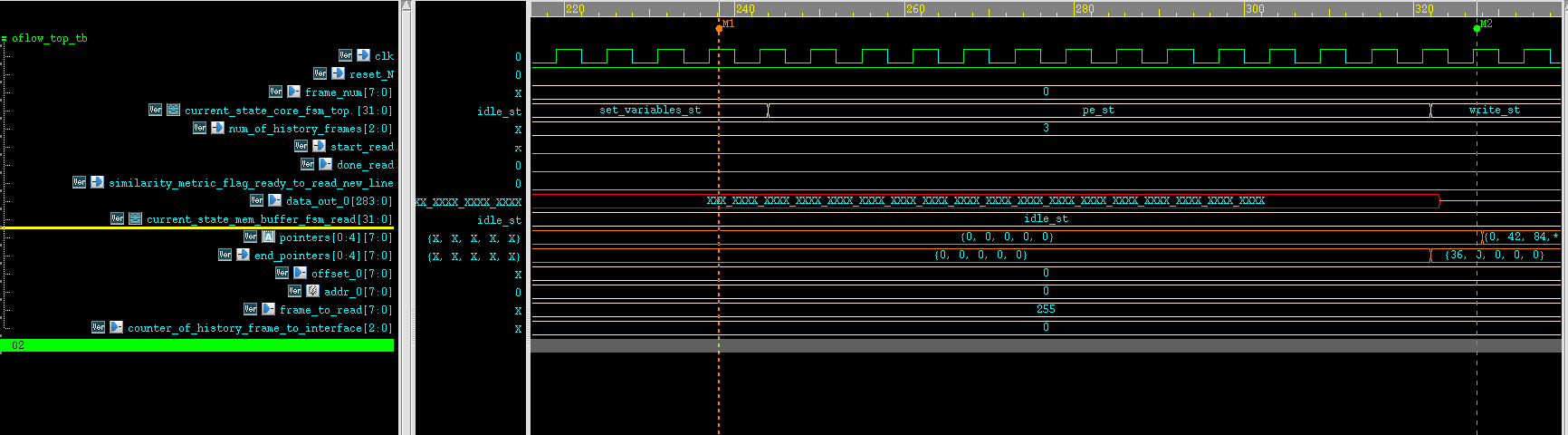
Frame 0:

Figure 43: oflow\_mem\_buffer\_wrapper – test 1 results - read - frame 0

As mentioned in pass criteria in the above table, there is no reading process in this frame.

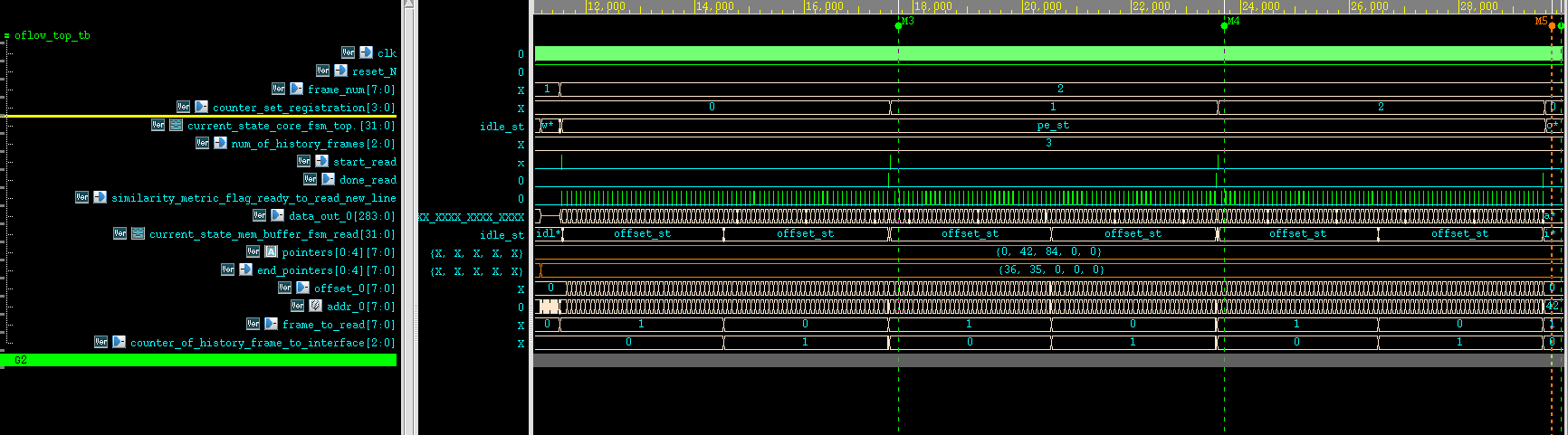
Frame 2:

Figure 44: oflow\_mem\_buffer\_wrapper - test 1 results - write - frame 2

As mentioned in pass criteria in the above table, start\_read and done\_read are raised in each set. In addition, similarity\_metric\_flag\_read\_new\_line raised 71 times for each set. frame\_to\_read is 1,0 for each set as expected. Moreover, the current\_state of the mem\_buffer\_fsm\_read switches between offset\_st and frame\_st in each set and return to idle between the different sets.

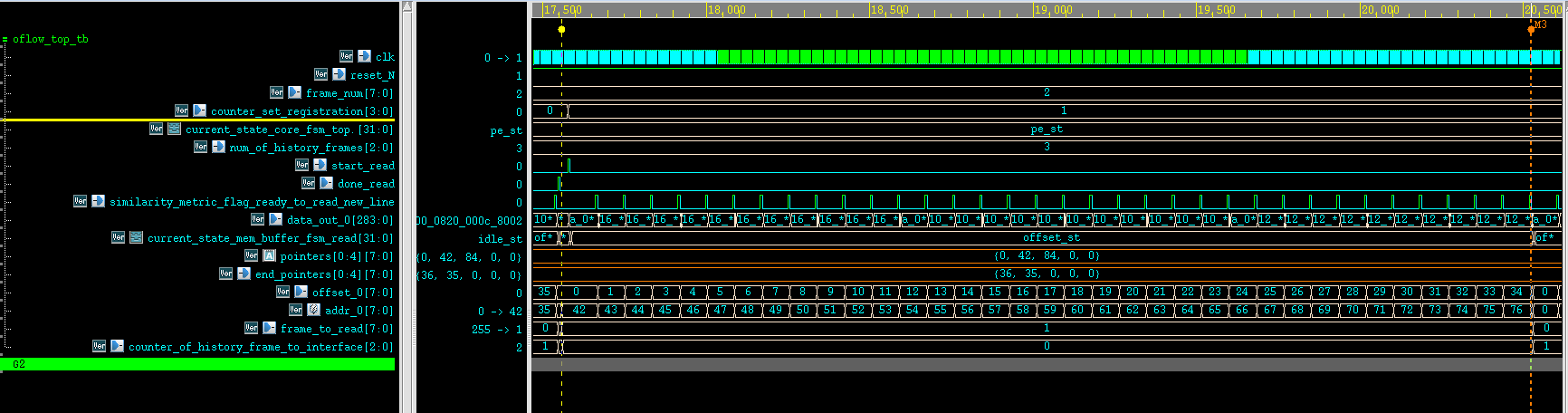


Figure 45: oflow\_mem\_buffer\_wrapper - test 1 results - read - frame 2 - set 1 - frame\_history=1

As mentioned in pass criteria in the above table, for set 1 of frame 2, addr\_0 is strating from 42 and ending in 76 (There are 70 BBoxes in frame 1 🡪 35 rows 🡪 [42,76] addresses).

# 4. Summary

## 4.1 Project's summary

## 

* In Summary, the goal of the project is to accelerate, by scalable HW, the object registration process.
* Innovation: Original and new HW algorithm, especially Conflict Resolve unit.
* It was achieved by planning a micro-architecture with units that work in parallel (multiple parallel process engine units, multiple ports for read and write transaction and storing of more than one BBox in one row of the memory).
* The main stages of the model are features extraction, similarity calculation, conflict resolve and data storage.
* What has been done in this part of the project?
  + Micro-Architecture (Blocks Diagram)
  + Implementation of all the units
  + Tests and functional simulations of part A units
* What still needs to be done in part B of the project**?**
  + Tests and simulations of the other units
  + Synthesis
  + performance
  + layout

## 4.2 Take message home

* Planning Micro-Architecture involves deep thinking.
* Correct implementation and tests in SV
* Learning new topics, such as fixed-point representation, fixing overflow, how to work with IPs and SRAM (their interface and the way they behave).

## 4.3 Next steps

* Raising the number of the Process Engines to 32 (power of 2), to accelerate the calculation and therefore the total process.
* Raising the number of fallbacks for the oflow\_conflict\_resolve to 5 instead of 2 for precision in the labeling of the BBoxes.
* Enlarge the similarity metric units per PE, to accelerate the process of the registration of the BBoxes.
* Enlarge the number of ports of memory access, to accelerate the process of the registration and the write stages.
* Remove the dependency in the glue logic and work directly with the DMA.

# 5.References

* 1. Optical Flow - <https://en.wikipedia.org/wiki/Optical_flow>
* 2. AMBA APB - https://developer.arm.com/documentation#numberOfResults=48&q=Amba%20apb
* 3. Article: Liu, Min, and Tobi Delbruck. "Block-matching optical flow for dynamic vision sensors: Algorithm and FPGA implementation." 2017 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2017.
* 4. Article – Sport-Analytics with YOLO et al. by Shahar Gino: https://medium.com/p/951b3f26221b
* 5. fixed-point representation: <https://youtu.be/b57jNWbqdmM?si=I5uaxTGcMJB4dMXZ>, https://projectf.io/posts/fixed-point-numbers-in-verilog/