



ARQUITECTURA DE ALTAS PRESTACIONES PARA SISTEMAS DE VISIÓN BIOINSPIRADOS

Arquitectura de Altas Prestaciones para Visión

2021

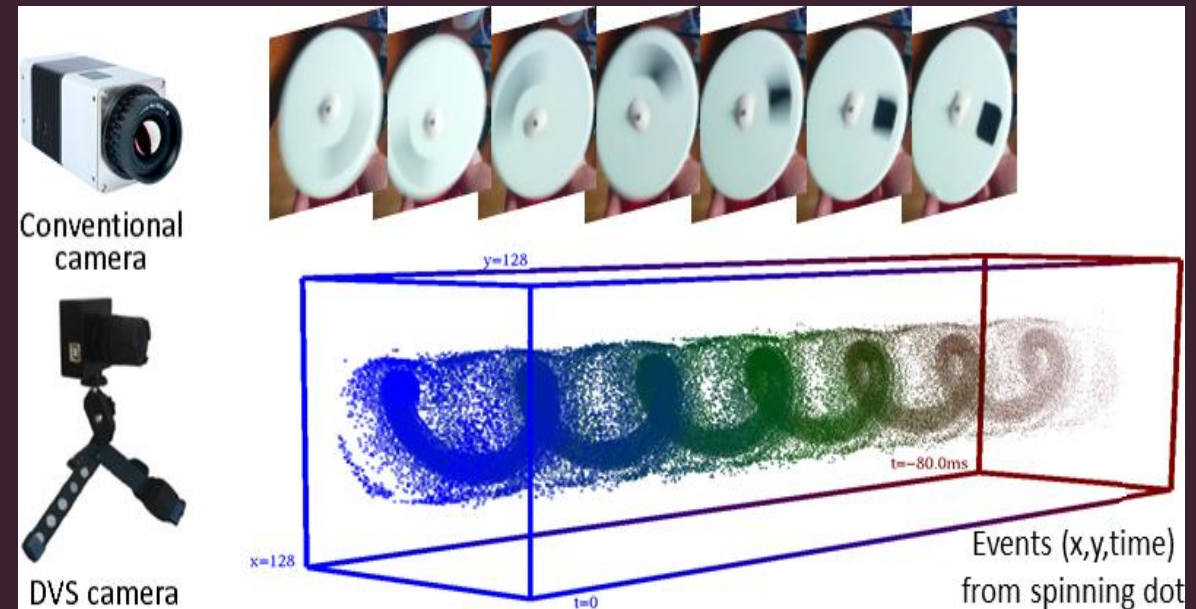
Guillermo José Fernández Caballero

ÍNDICE

- Contexto
- Problema a solucionar
- Motivación
- Estado del arte
- Implementación
- Trabajo futuro

CONTEXTO. CÁMARA DE EVENTOS.

- Funcionamiento similar al de la retina.
- Registra eventos relacionados con cambios en la luminosidad de la escena. Funcionamiento asíncrono.
- Si nada cambia, no se registra información.
- Cuando un pixel cambia de luminosidad, manda una señal de salida que contiene:
 - *coordenadas del píxel.*
 - *marca de tiempo.*
 - *polaridad (opc.).*
 - *nivel de brillo (opc.).*



CONTEXTO. CÁMARA DE EVENTOS.

VENTAJAS

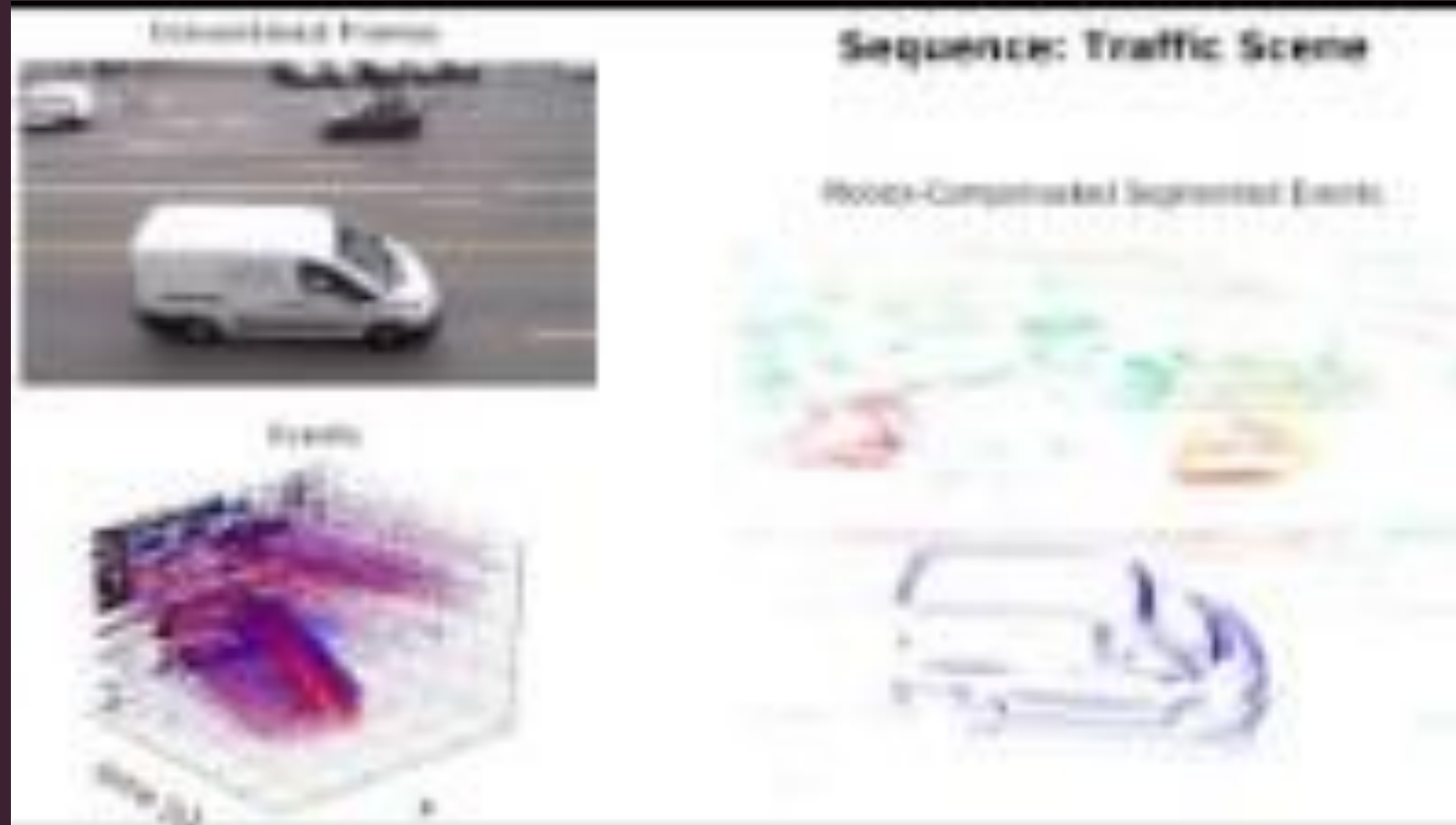
- Alta resolución temporal (μ s).
- Baja latencia.
- No hay desenfoque por movimiento.
- Bajo consumo de energía (30 mW).
- Alto rango dinámico (> 120 dB).

INCONVENIENTES

- Funcionamiento asíncrono (rediseño de algoritmos de procesamiento).
- Resolución (0,1 – 0,2 MPx).
- Precio elevado (1.000 € - 5.000 €).

Sensor	Dynamic range (dB)	Equivalent framerate* (fps)	Spatial resolution (MP)	Power consumption (mW)
Human eye	30–40	200-300	-	10 ^[4]
High-end DSLR camera (Nikon D850)	44.6 ^[5]	120	2–8	-
Ultrahigh-speed camera (Phantom v2640) ^[6]	64	12,500	0.3–4	-
Event camera ^[7]	120	1,000,000	0.1–0.2	30

CONTEXTO. CÁMARA DE EVENTOS.



PROBLEMA A SOLUCIONAR

- Miles de eventos generados cada segundo.
- No sirven algoritmos tradicionales de procesamiento de imágenes.
- Necesidad de procesamiento instantáneo.
- Sistemas empotrados suelen tener poca capacidad de cálculo.
- Equipos potentes = gran espacio y gran consumo.



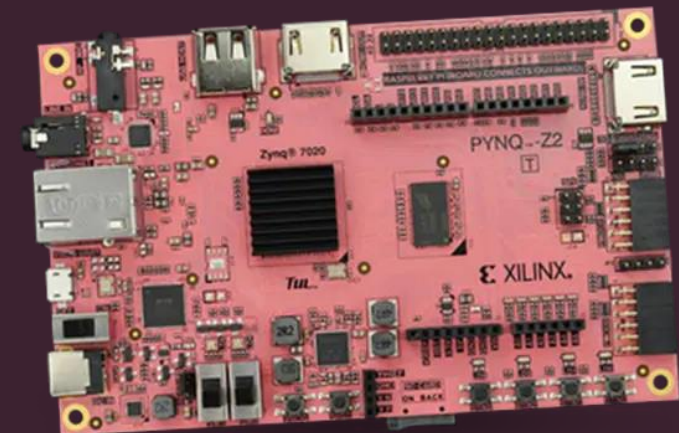
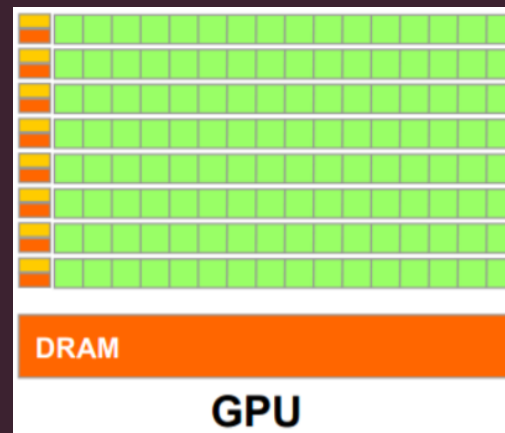
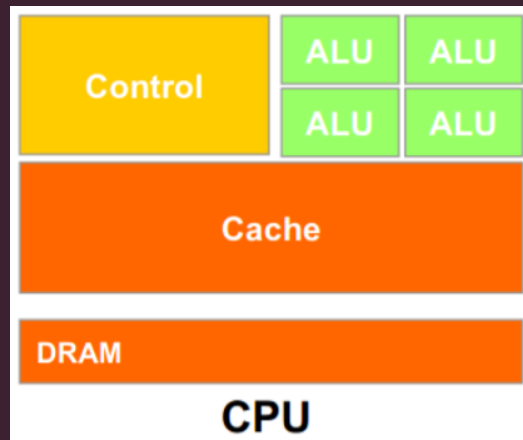
MOTIVACIÓN

- Utilización en temas de actualidad.
- Guiado de drones, conducción autónoma, sistemas tolerantes a fallos, etc.
- En lo personal, procesamiento de imágenes sumado a desarrollo hardware.
- Aplicación directa sobre TFM.



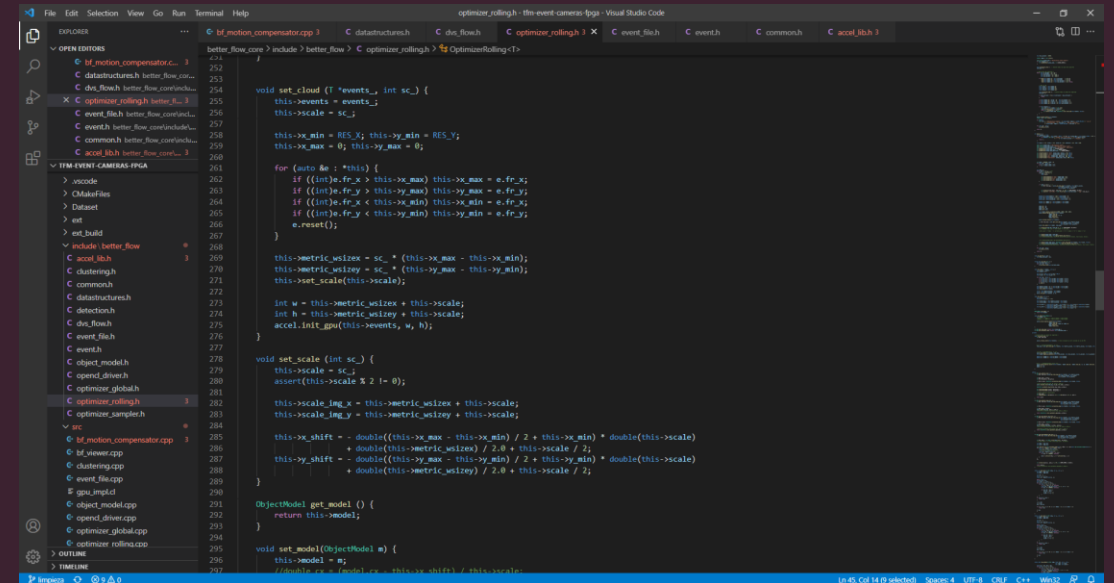
ESTADO DEL ARTE

- *Event-based Moving Object Detection and Tracking. Anton Mitrokhin et al. 2020.*
- Descripción e implementación del algoritmo en C++.
- Pruebas sobre CPU de PC, GPU de NVIDIA con CUDA y FPGA.



IMPLEMENTACIÓN

- Proyecto muy amplio (TFM).
- Adaptación:
 - Algoritmo principal compuesto por varias funciones.
 - Selección de una de estas funciones.
 - Implementación en Python.
 - Aceleración mediante FPGA.



```
void set_cloud (T *events, int sc_) {
    this->events = events;
    this->scale = sc_;

    this->xs_min = RES_X; this->ys_min = RES_Y;
    this->xs_max = 0; this->ys_max = 0;

    for (auto &e : *this) {
        if ((int)e.fr.x > this->xs_max) this->xs_max = e.fr.x;
        if ((int)e.fr.y > this->ys_max) this->ys_max = e.fr.y;
        if ((int)e.fr.x < this->xs_min) this->xs_min = e.fr.x;
        if ((int)e.fr.y < this->ys_min) this->ys_min = e.fr.y;
        e.reset();
    }

    this->metric_wsize_x = sc_ * (this->xs_max - this->xs_min);
    this->metric_wsize_y = sc_ * (this->ys_max - this->ys_min);
    this->set_scale(this->scale);

    int w = this->metric_wsize_x + this->scale;
    int h = this->metric_wsize_y + this->scale;
    accel.init_gpu(this->events, w, h);
}

void set_scale (int sc_) {
    this->scale = sc_;
    assert(this->scale > 0);

    this->scale_img_x = this->metric_wsize_x + this->scale;
    this->scale_img_y = this->metric_wsize_y + this->scale;

    this->xs_shift = - double((this->xs_max - this->xs_min) / 2 + this->xs_min) * double(this->scale)
        + double(this->metric_wsize_x) / 2.0 + this->scale / 2;
    this->ys_shift = - double((this->ys_max - this->ys_min) / 2 + this->ys_min) * double(this->scale)
        + double(this->metric_wsize_y) / 2.0 + this->scale / 2;
}

ObjectModel get_model () {
    return this->model;
}

void set_model(ObjectModel m) {
    this->model = m;
    //double xs = (double)m.x - this->xs_shift; // this->scale;
```

IMPLEMENTACIÓN

- Algoritmo 1:
 - *get_time_img()*
 - *Sobel_cpu()*
 - *Update_accumulators()*
 - *Project_4param_reinit()*

Algorithm 1 Global motion compensation in event space using \mathcal{T} .

Data: $\mathcal{M}_{i-1}^G, C, d, \xi$

Result: $\mathcal{M}_i^G, C', \mathcal{T}$

```
 $C' \leftarrow \text{warpEventCloud}(C, \mathcal{M}_{i-1}^G)$   
 $\mathcal{T} \leftarrow \text{getTimestampImage}(C', d)$   
 $\mathcal{M}_i^G \leftarrow \text{updateModel}(\mathcal{M}_{i-1}^G, \mathcal{T})$   
while  $\|\mathcal{M}_{i-1}^G - \mathcal{M}_i^G\|_2 > \xi$  do  
   $C' \leftarrow \text{warpEventCloud}(C, \mathcal{M}_i^G)$   
   $\mathcal{T} \leftarrow \text{getTimestampImage}(C', d)$   
   $\mathcal{M}_{i-1}^G \leftarrow \mathcal{M}_i^G$   
   $\mathcal{M}_i^G \leftarrow \text{updateModel}(\mathcal{M}_i^G, \mathcal{T})$   
end
```

```
/** T <- getTimestampImage(C', d) */  
time_img = accel.get_time_img(this->events, this->metric_wsize, this->metric_hsize, this->scale, this->x_shift, this->y_shift); // Obtiene la time_image  
  
/** Mi <- updateModel(Mi-1, T) */  
accel.fast_model(this->model, time_img);  
this->model.update_accumulators(this->rot_divider, this->div_divider, this->x_divider, this->y_divider);  
  
double cx = (model.cx - this->x_shift) / this->scale;  
double cy = (model.cy - this->y_shift) / this->scale;  
  
/** C' <- warpEventCloud(C, Mi-1) */  
accel.project_4param_reinit<T>(this->events, -model.total_dx, -model.total_dy, cx, cy, model.total_div, -model.total_rot);  
model.cx = cx;  
model.cy = cy;
```

IMPLEMENTACIÓN

- *Get_time_img()*:
 - Obtiene la imagen temporal a partir de un array de eventos.
 - Esta imagen contiene en cada pixel la media de los timestamps de los eventos mapeados en dicho pixel.

Eventos

```
events.txt
events.txt
1 1519939777.722010347 8 58 1
2 1519939777.722010347 114 59 1
3 1519939777.722010347 102 57 1
4 1519939777.722010347 117 67 1
5 1519939777.722014347 56 70 0
6 1519939777.722014347 147 74 0
7 1519939777.722014347 32 74 1
8 1519939777.722025347 142 77 0
9 1519939777.722025347 115 77 0
10 1519939777.722026347 91 76 0
11 1519939777.722026347 1 91 1
12 1519939777.722044347 108 82 0
13 1519939777.722044347 84 82 0
14 1519939777.722047347 146 84 0
15 1519939777.722051347 164 112 1
16 1519939777.722051347 175 113 1
17 1519939777.722051347 154 113 1
18 1519939777.722053347 173 114 1
19 1519939777.722054347 158 116 1
20 1519939777.722054347 157 118 1
21 1519939777.722054347 171 124 1
22 1519939777.722057347 159 125 1
23 1519939777.722058347 154 127 1
24 1519939777.722058347 168 121 1
25 1519939777.722075347 154 129 1
26 1519939777.722079347 121 130 0
```

Código en Python

```
def get_time_img_cpu2(eventos, w, h, scale, x_sh, y_sh):
    w = int(w)
    h = int(h)
    scale = int(scale)
    x_sh = int(x_sh)
    y_sh = int(y_sh)

    project_img_avg = np.zeros((w + scale, h + scale), dtype = np.single)
    project_img_cnt = np.zeros((w + scale, h + scale), dtype = np.single)
    project_img_res = np.zeros((w + scale, h + scale), dtype = np.single)

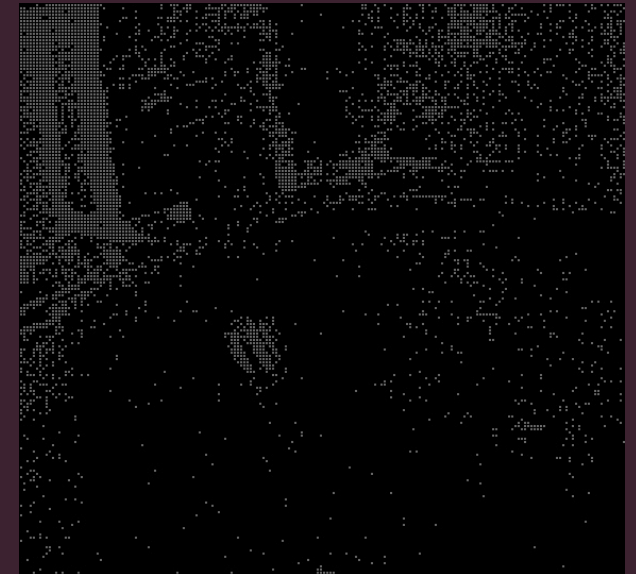
    for e in eventos:
        x = e.x * scale + x_sh
        y = e.y * scale + y_sh

        project_img_avg[x, y] = project_img_avg[x, y] + float(e.t) / 1000000000.0
        project_img_cnt[x, y] = project_img_cnt[x, y] + 1
        tmp = project_img_avg[x, y] / project_img_cnt[x, y]

        for jx in range((x - int(scale / 2)), (x + int(scale / 2))):
            for jy in range((y - int(scale / 2)), (y + int(scale / 2))):
                project_img_res[jx, jy] = tmp

    return project_img_res
```

Time image



IMPLEMENTACIÓN

- Aceleración utilizando Overlays personalizados con Pynq Z2.
- Vitis HLS para crear IP personalizado para acelerar.
- Vivado para crear la arquitectura del Overlay.
- Python para ejecutar el algoritmo en la Pynq Z2.



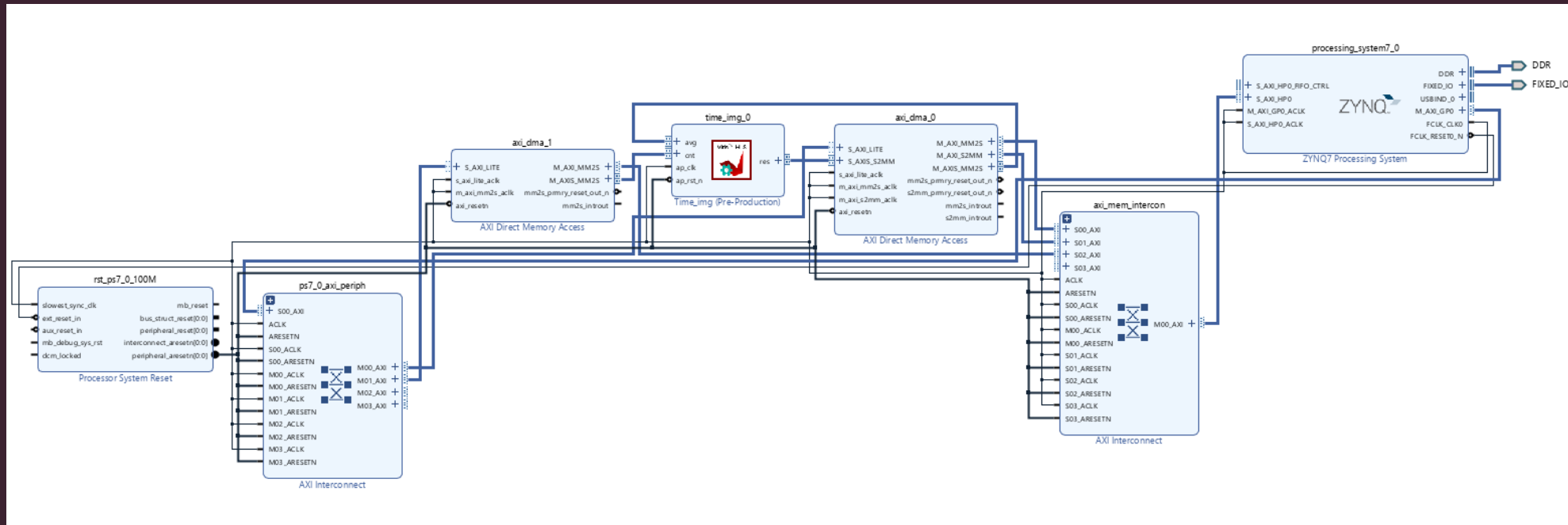
IMPLEMENTACIÓN

- Muchos problemas en el proceso.
- Pocos tutoriales realmente buenos.
- Solo implementaciones sencillas y sin explicaciones.
- Demasiada (des)información en el foro de Xilinx.
- Desarrollo hardware a bajo nivel es una tarea complicada.

No se llega a ninguna implementación válida

IMPLEMENTACIÓN

```
aapv.cpp x Synthesis Summary(solution1)
1 void time_img(float avg[1024][1024], int cnt[1024][1024], float res[1024][1024])
2
3     for(int i = 0; i < 1024; i++){
4         for(int j = 0; j < 1024; j++){
5             res[i][j] = avg[i][j] / cnt[i][j];
6         }
7     }
8
9 }
10
```



IMPLEMENTACIÓN

```
In [38]: # Importa el overlay creado (La carpeta debe incluir el .bit, .hwh y .tcl con el mismo nombre)

from pynq import Overlay
import numpy as np
import pynq.lib.dma

aapvol = Overlay('/home/xilinx/pynq/overlays/aapv/aapv.bit')

dma0 = aapvol.axi_dma_0
dma1 = aapvol.axi_dma_1
```

```
In [39]: from pynq import Xlnk
xlnk = Xlnk()
```

```
In [40]: avg = xlnk.cma_array(shape=(32,32), dtype=np.float)
cnt = xlnk.cma_array(shape=(32,32), dtype=np.int)
res = xlnk.cma_array(shape=(32,32), dtype=np.float)

for i in range(32):
    for j in range(32):
        avg[i][j] = 8.0;
        cnt[i][j] = 2;
```

```
In [41]: dma0.sendchannel.transfer(avg)
dma1.sendchannel.transfer(cnt)
dma0.recvchannel.transfer(res)
```

Repositorio GitHub:

<https://github.com/GuilleFdez/AAPV>

```
In [33]: print(res)

[[0.3125 0.3125 0.3125 ... 0.3125 0.3125 0.3125]
 [0.3125 0.3125 0.3125 ... 0.3125 0.3125 0.3125]
 [0.3125 0.3125 0.3125 ... 0.3125 0.3125 0.3125]
 ...
 [0.      0.      0.      ... 0.      0.      0.      ]
 [0.      0.      0.      ... 0.      0.      0.      ]
 [0.      0.      0.      ... 0.      0.      0.      ]]
```

DEMOSTRACIÓN

Código Python sin acelerar pero optimizado

TRABAJO FUTURO

- Obtener el núcleo del código en C++.
- Acelerar aquellas partes que sea posible en hardware.
- Creación de IP's en Vitis HLS.
- Creación del layout en Vivado.
- Programación de la Pynq Z2 en C++ usando Vitis.

PREGUNTAS

Muchas gracias.