Progetto Smart Camera per Rimozione dello Sfondo

Esame di Sistemi Digitali M

Riccardo Gandolfi - 0000901940
Alessandro Nadalini - 0000901504

A.A 2020/2021

Indice

1: Introduzione

Il presente progetto si configura come un possibile metodo per elaborare immagini ed effettuare l'eliminazione dello sfondo al fine di isolare figure e soggetti inquadrati dal sensore camera OV7670. Questo sistema può trovare applicazione in sistemi di videosorveglianza.

La strumentazione utilizzata consiste in una camera OV7670, una Digilent Zedboard e un monitor provvisto di ingresso VGA (risoluzione 640x480). Il funzionamento del sistema è il seguente:

- Tramite pressione di un pulsante viene immagazzinato in memoria un frame di riferimento (la modalità di acquisizione verrà illustrata in seguito).

Il comportamento successivo del sistema è controllato da uno switch connesso ad un segnale di enable (attivo alto):

- Enable=0: il frame mostrato a schermo è quello attualmente visto dalla camera, scritto in memoria e inviato sotto forma di AXI stream all'interfaccia VGA. In questo caso, nessuna forma di elaborazione viene effettuata sul frame in arrivo dalla camera.
- Enable=1: il frame mostrato a schermo è il risultato del confronto tra il frame acquisito dalla camera e quello di riferimento immagazzinato in memoria. Il confronto, come verrà illustrato in seguito, consiste a grandi linee nel realizzare una differenza pixel a pixel tra i due frame (gestiti sotto forma di stream AXI a 8 bit). Ovviamente, tale operazione rende necessaria l'implementazione di un meccanismo di sincronizzazione tra i due AXI stream, al fine di elaborare sempre pixel corrispondenti alla stessa posizione nei due frame.

Di seguito, viene illustrato in dettaglio il sistema realizzato. Innanzitutto, verrà presentata la board utilizzata, le connessioni utilizzate per il sensore video e il mapping dei pin di I/O, necessari all'implementazione dello switch di enable e il pulsante collegato all'interrupt.

In secondo luogo, verrà illustrato il block design realizzato su Vivado, per poi proseguire con la descrizione dei vari moduli realizzati tramite Vivado HLS (in linguaggio C++) e Vivado (in linguaggio Verilog). In particolare, ne verranno discusse l'interfaccia, il funzionamento e il codice scritto.

Dopodiché, saranno illustrate e discusse le modifiche apportate al codice C già esistente in Vivado SDK al fine di programmare il processore Zynq per effettuare il salvataggio in memoria del frame di riferimento.

Infine, verranno mostrati i risultati sperimentali ottenuti e saranno tratte le debite conclusioni.

2: Zedboard: Layout e I/O Mapping

Per quanto riguarda la connessione del sensore video alla board, sono stati utilizzati i PMOD A e B della Zedboard. Oltre a questi, sono stati utilizzati uno switch connesso al segnale di enable per effettuare l'operazione di confronto e uno dei pulsanti per realizzare il meccanismo a interrupt di acquisizione del frame di riferimento.

Innanzitutto, il sensore OV7670 è stato connesso come segue:

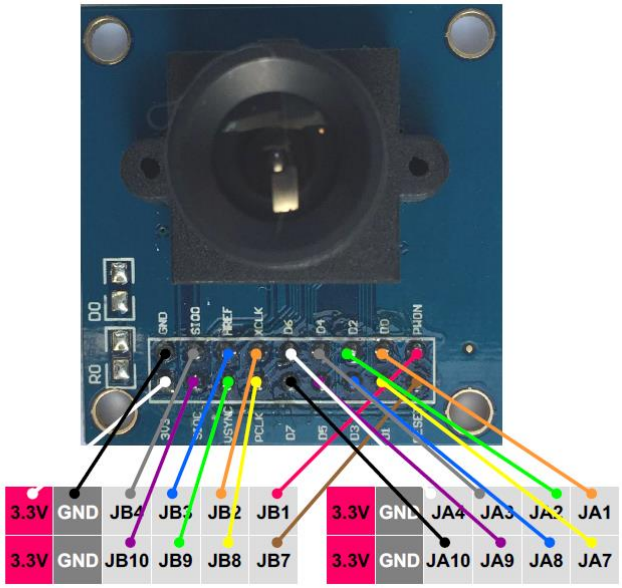

Figura 1: OV7670 e connessioni ai PMOD della Zedboard

I segnali di enable del confronto e di acquisizione del frame di riferimento sono stati mappati sui pin seguenti:

Segnale	Direzione	Package pin	I/O Std
en_compare	IN	H22	LVCMOS33
capture_ref	IN	P16	LVCMOS33

Il mapping dei pin restanti non è stato modificato rispetto al progetto di partenza presente su Github (https://github.com/stefano-mattoccia/SmartCamera).

Di seguito vengono illustrati il block design realizzato su Vivado e il funzionamento dettagliato del sistema.

3: Vivado Block Design e Funzionamento

Il block design complessivo del sistema è il seguente:

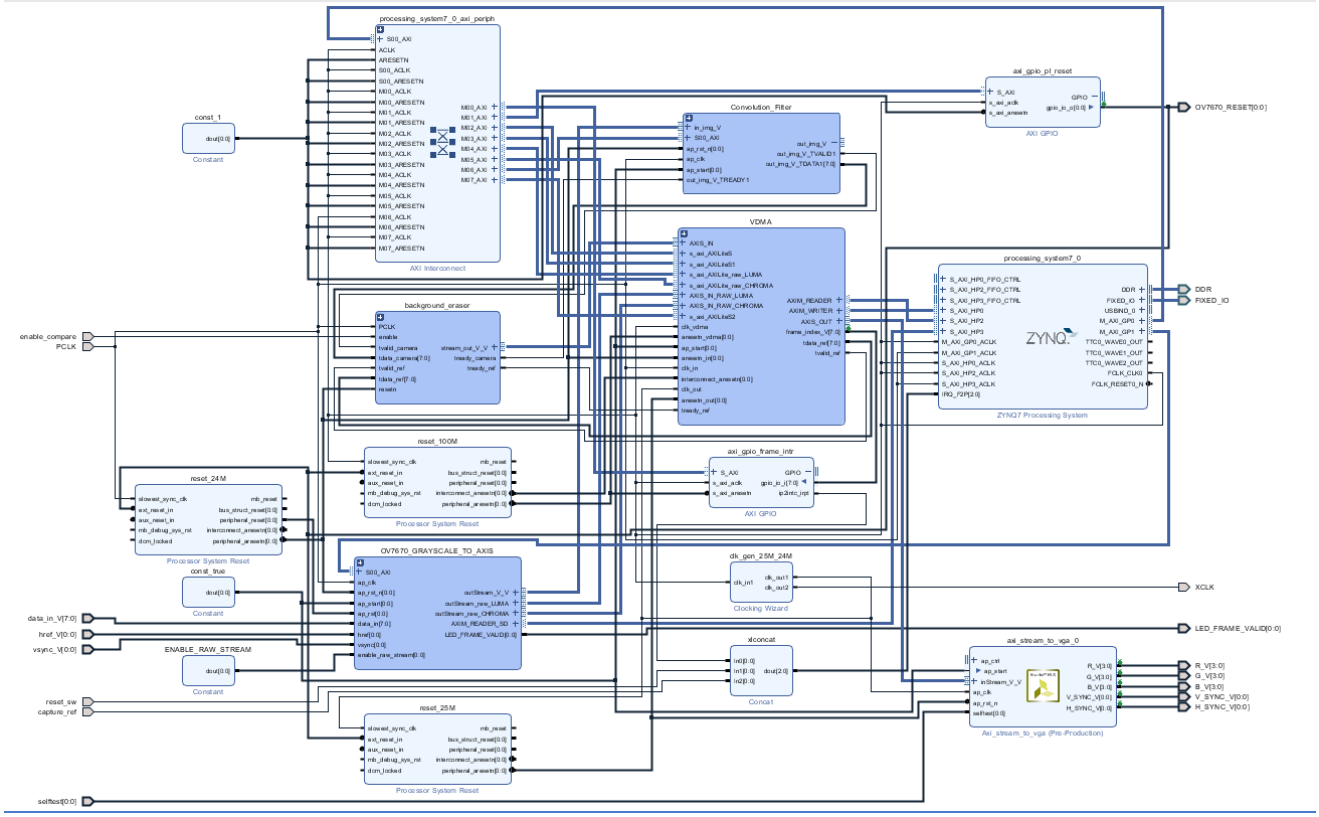

Figura 2: Vivado Block Design

Le modifiche apportate al block design del progetto originale sono elencate di seguito:

- Gerarchia di moduli denominata *Background_eraser*.
- Un *ddr_to_axis_reader* e relativi moduli di interfacciamento all'interno del VDMA.
- Ingressi aggiuntivi di enable e interrupt per acquisizione del frame di riferimento.

Il funzionamento del sistema è incentrato sulla gerarchia *Background_eraser,* al suo interno sono infatti implementati il meccanismo di sincronizzazione degli stream e il modulo che effettua l'elaborazione dell'immagine controllata dal segnale di enable. Di seguito sono illustrati il block design della gerarchia e le diverse fasi di funzionamento.

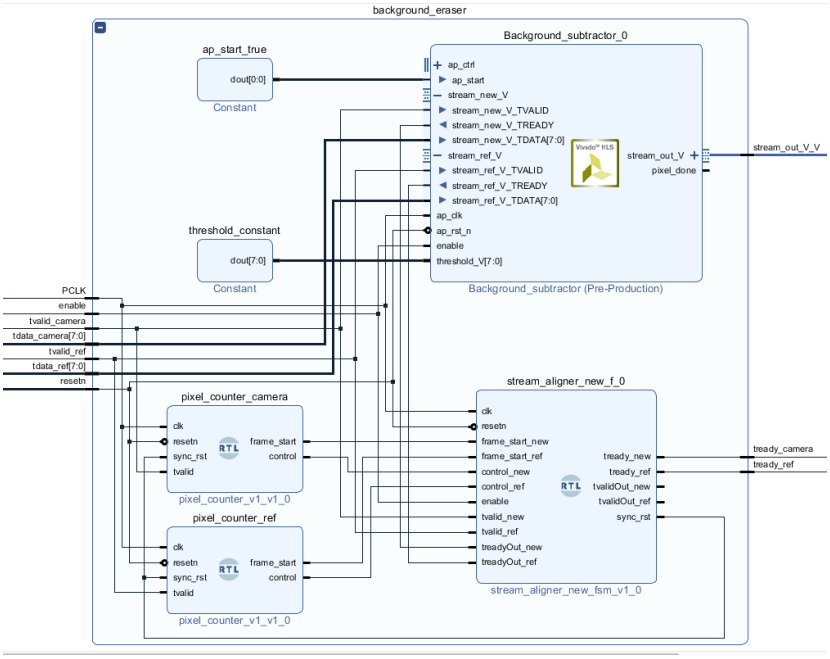

Figure 3: Block design della gerarchia Background_eraser

Enable_compare=0/No interrupt

Lo stream acquisito dalla camera attraversa dei blocchi di interfaccia e un filtro di convoluzione fino all'ingresso del *Background_eraser*. All'interno della gerarchia di blocchi, questo stream viene lasciato passare senza alcuna elaborazione all'interno del *Background_subtractor* per poi essere inviato al VDMA al fine di effettuarne lo storage in memoria ddr. Dunque, durante questa fase, né il meccanismo di sincronizzazione né quello di elaborazione effettuano alcuna operazione, se non quella di lasciar passare lo stream in modo trasparente.

Enable_compare=0/Arrivo dell'interrupt

All'arrivo dell'interrupt, la cui gestione via software sarà discussa nella sezione relativa alle modifiche apportate su Vivado SDK, il frame attualmente acquisito dalla OV7670 viene immagazzinato in memoria ad un determinato indirizzo. Al fine di effettuarne il confronto con il frame acquisito dalla camera, l'uscita del *ddr_to_axis_reader* aggiuntivo viene retroazionata in ingresso alla gerarchia *Background_eraser* e, in particolare, all'ingresso del modulo *Background_subtractor*. Dunque, sono questi i due AXI stream sui quali agiscono i meccanismi di sincronizzazione ed elaborazione una volta settato a 1 l'ingresso di enable.

Enable_compare=1/No interrupt

Dopo aver acquisito il frame di riferimento, è possibile settare a 1 il segnale di enable del confronto, innescando dunque i meccanismi di sincronizzazione ed elaborazione dei frame. Mentre questi meccanismi saranno descritti in dettaglio nelle sezioni relative ai moduli HLS ed

RTL, è comunque possibile descriverne a grandi linee il funzionamento. Il meccanismo di sincronizzazione è stato realizzato impiegando due moduli realizzati in linguaggio Verilog:

- *Pixel_counter:* questo modulo realizza un contatore di pixel, necessario per determinare correttamente l'inizio del frame di ciascuno stream. A tal fine sono implementati due segnali di controllo:
    - *Frame_start:* mantenuto a 1 per un solo ciclo di clock ogni 640x480 cicli di conteggio.
    - *Frame_control:* posto a 1 dopo l'asserzione di *Frame_start* e mantenuto a questo valore fintanto che il segnale *t_valid* del relativo stream resta alto.
- *Stream_aligner:* sfruttando i segnali del protocollo AXI stream e quelli prodotti dal *Pixel_counter*, questo modulo effettua la sincronizzazione dei due stream in ingresso al *Background_subtractor* solo quando il segnale di enable è settato a 1. In caso contrario il modulo risulta trasparente ad entrambi gli stream. Il meccanismo di sincronizzazione è basato su una FSM il cui funzionamento verrà analizzato in dettaglio nella sezione dedicata.

Dopo aver sincronizzato i due stream, spetta al modulo *Background_subtractor* effettuare l'elaborazione dell'immagine. L'elaborazione viene effettuata tramite una differenza pixel a pixel tra i due frame per identificare se siano uguali oppure no:

- Quando la differenza tra due pixel risulta minore di una soglia impostata dall'esterno, essi vengono assunti uguali, dunque il pixel risultante in uscita viene settato a 0xFF.
- Quando la differenza tra due pixel risulta maggiore di questa soglia, essi vengono assunti diversi. Il pixel risultante in uscita sarà posto uguale a quello restituito dal filtro di convoluzione a monte.

Lo stream così realizzato in uscita dal *Background_eraser* viene inviato al VDMA per effettuarne lo storage in memoria ddr. La relativa uscita del VDMA è connessa al modulo *axis_to_VGA* per la visualizzazione sul monitor, mentre l'altra uscita, come accennato in precedenza, è retroazionata in ingresso al modulo *background_subtractor*.

Dunque, il funzionamento del sistema è basato sull'assunzione che, prima di innescare l'elaborazione dell'immagine, un frame di riferimento venga acquisito e immagazzinato in memoria al fine di effettuare l'operazione di confronto. Analogamente, si suppone che il pulsante per l'acquisizione del riferimento non venga premuto a elaborazione in corso, dunque con switch di enable settato a 1. Di fatto, l'acquisizione del frame di riferimento durante l'elaborazione dell'immagine porterebbe allo storage di un frame errato, ovvero quello soggetto alla rimozione dello sfondo piuttosto che un frame da utilizzare come background.

Nelle sezioni successive saranno descritti i moduli implementati, iniziando con quello realizzato in C++ tramite Vivado HLS per poi proseguire con quelli realizzati in Verilog direttamente su Vivado.

4: Moduli HLS

Vivado HLS è stato utilizzato per realizzare il modulo di elaborazione *Background_subtractor*. Di seguito sono riportati i file *Background_subtractor.cpp* e *Background_subtractor.h*:

```
#include "Background_subtractor.h"

void  Background_subtractor (byte stream_new[FRAMESIZE], byte stream_ref[FRAMESIZE], volatile bool *enable, byte *threshold, volatile bool *pixel_done, byte stream_out[FRAMESIZE])
{

    #pragma HLS INTERFACE ap_none port=enable
    #pragma HLS INTERFACE ap_none port=threshold
    #pragma HLS INTERFACE axis port=stream_new
    #pragma HLS INTERFACE axis port=stream_ref
    #pragma HLS INTERFACE axis port=stream_out
    #pragma HLS INTERFACE ap_none port=pixel_done

    byte pixel_new, pixel_ref;
    int diff;
    int i, cont, j;
    bool int_done=false;


    for(j=0; j<HEIGHT; j++)
        for(i=0; i<WIDTH; i++)
        {

            pixel_new = stream_new[j*WIDTH+i];
            pixel_ref = stream_ref[j*WIDTH+i];

            if (*enable)
                {
                    diff=(int)(pixel_new-pixel_ref);
                    if (abs(diff)<(*threshold))
                    {
                        stream_out[j*WIDTH+i] = 0xff;
                    }
                    else
                    {
                        stream_out[j*WIDTH+i] = stream_new[j*WIDTH+i];
                    }
                }
            else
            {
                stream_out[j*WIDTH+i] = stream_new[j*WIDTH+i];
            }

            /* This flag tells us that a single pixel has been processed. */

            int_done = !int_done;
            *pixel_done=int_done;

        }
    return;
}
```

Figura 4: Background_subtractor.cpp

```
#define WIDTH 640
#define HEIGHT 480
#define FRAMESIZE WIDTH*HEIGHT

typedef ap_uint<8> byte;

void  Background_subtractor (byte stream_new[FRAMESIZE], byte stream_ref[FRAMESIZE], volatile bool *enable, volatile bool *pixel_done, byte stream_out[FRAMESIZE]);
```

Figura 5: Background_subtractor.h

Osservando l'interfaccia del modulo:

- *stream_new:* vettore di 640x480 pixel (definiti come ap_uint<8>) proveniente dal filtro di convoluzione. Dunque, questo stream contiene il frame attualmente acquisito dalla camera.
- *stream_ref:* vettore analogo a *stream_new*, con la differenza che proviene dal VDMA. Questo stream contiene infatti il frame di riferimento immagazzinato in memoria all'arrivo della relativa interrupt.
- *enable:* questo segnale, definito come volatile bool, determina il comportamento del modulo. Quando è uguale a 1, la rimozione dello sfondo viene innescata, altrimenti il modulo risulta trasparente allo stream proveniente dal filtro di convoluzione a monte.
- *pixel_done:* questo flag viene utilizzato per notificare l'avvenuta elaborazione, o il semplice passaggio, di un pixel dello stream che verrà poi trasmesso in uscita.
- *stream_out:* questo stream, definito in modo analogo a quelli di ingresso, è l'uscita principale del modulo *Background_subtractor*. A seconda del segnale di enable, contiene i pixel frutto dell'elaborazione tra i due stream in ingresso, oppure il pixel attualmente inviato dal filtro di convoluzione. Questo stream di uscita viene poi inviato in ingresso al VDMA per il suo storage in ddr.

- *threshold:* segnale a 8 bit che determina la soglia di confronto tra due pixel.

Le pragma utilizzate in HLS sono l'axis_port per gli stream e l'ap_ctrl_none per i segnali di enable, threshold e pixel_done.

Clock e reset

Questo modulo riceve il segnale di clock dal pin connesso al PCLK della OV7670 (clock a 24 MHz). Il segnale di reset è anch'esso a 24 MHz. Ciò si rende necessario dal momento che l'elaborazione viene effettuata alla frequenza del clock della camera.

Funzionamento

Come illustrato in precedenza, il comportamento di questo modulo è determinato dal segnale di enable mappato sullo switch H22. Nel caso in cui tale segnale sia settato a 1, l'elaborazione dei due stream in ingresso viene innescata. Tale operazione, come accennato in precedenza, è basata su una differenza tra i due pixel letti in ingresso, la quale viene poi confrontata in valore assoluto con una soglia determinata da un blocco costante su Vivado. Dunque, due pixel la cui differenza risulta minore di questa soglia sono assunti uguali e settati a 0xFF al fine di eliminare lo sfondo dell'immagine e sostituirlo con un background bianco. In caso contrario, il pixel dello stream in uscita viene posto uguale a quello attualmente ricevuto dal filtro di convoluzione.

5: Moduli Verilog

I moduli realizzati in linguaggio Verilog direttamente in ambiente Vivado sono il *pixel_counter* e lo *stream_aligner.* Entrambi i moduli saranno adesso analizzati e discussi in dettaglio.

Pixel Counter

All'interno della gerarchia *Background_eraser* sono stati inseriti due pixel counter, ciò è necessario al fine di implementare il meccanismo di sincronizzazione degli stream. Di seguito viene illustrato il codice Verilog del contatore:

```
`timescale 1ns / 1ps

module pixel_counter_v1
(
    input wire clk,
    input wire resetn,
    input wire sync_rst,
    input wire tvalid,
    output reg frame_start,
    output reg control
);

reg end_count,flag;
reg [18:0] count_value;
reg en_count;

// State variables
parameter [1:0] idle=2'd0, count=2'd1, frame_sync=2'd2;
reg [1:0] cs;
reg [1:0] ns;

always@(posedge clk, negedge resetn, posedge sync_rst)
begin
    if (resetn==1'b0 || end_count==1'b1 || sync_rst==1'b1)
        count_value<=19'b0;
    else
    begin
        if (en_count==1'b1)
            count_value<=count_value+19'b1;
        else
            count_value<=count_value;
    end
end

always@(*)
begin
    if (count_value==19'h4afff)
        end_count=1'b1;
    else
        end_count=1'b0;
end
```

Figure 6: pixel_counter_v1.v

```
// Sequential FSM
always@(posedge clk, negedge resetn)
begin
    if (resetn==1'b0)
        cs<=idle;
    else
        cs<=ns;
end

// Combinational FSM
always@(*)
begin
frame_start=1'b0;
en_count=1'b0;

    case (cs)

    idle:
    begin
        flag=1'b0;
        if (tvalid==1'b1)
            ns=count;
        else
            ns=cs;
    end

    count:
    begin
        if (tvalid==1'b0)
            ns=idle;
        else
        begin
            en_count=1'b1;
            if (end_count==1'b0)
                ns=cs;
            else
                ns=frame_sync;
        end
    end

    frame_sync:
    begin
        flag=1'b1;
        frame_start=1'b1;
        if (tvalid==1'b1)
        begin
            ns=count;
            en_count=1'b1;
        end
        else
        begin
            ns=idle;
            en_count=1'b0;
        end
    end

    default:
    begin
        ns=idle;
        flag=1'b0;
        en_count=1'b0;
    end

    endcase
end

always@(*)
begin
    if (flag==1'b1)
    begin
        if (tvalid==1'b1)
            control=1'b1;
        else
            control=1'b0;
    end
    else
        control = 1'b0;
end

endmodule
```

Figure 7: pixel_counter_v1.v

In Fig.6 sono mostrati i segnali del modulo (sia d'interfaccia che interni), le variabili di stato e l'aggiornamento del conteggio. Sono di particolare interesse i seguenti segnali di interfaccia:

- *Sync_rst*: questo segnale viene posto a 1 dal modulo *Stream_aligner*. Ha lo scopo di resettare entrambi i pixel counter quando è necessario sincronizzare i due stream. In tal modo, il conteggio dei pixel dei due stream parte dal valore di reset.
- *Tvalid:* questo segnale fa parte del protocollo AXI stream e viene utilizzato per controllare la presenza di un dato valido in ingresso. All'interno del modulo *Pixel_counter* viene utilizzato per aggiornare il segnale *control* e per abilitare il

conteggio. Queste funzionalità sono realizzate nella macchina a stati e nel blocco always@ illustrati in Fig.7.

- *Frame_start:* questo segnale viene utilizzato per segnalare il primo pixel di un frame dello stream. A differenza del segnale *control, frame_start* resta alto per un solo ciclo di clock. Si noti che è un segnale di uscita, infatti, viene inviato al modulo *Stream_aligner,* dove sarà utilizzato per controllare lo stato dei due AXI stream in ingresso alla gerarchia.
- *Control:* questo segnale viene settato a 1 dopo l'asserzione di *frame_start* e continuamente aggiornato controllando il segnale *tvalid*. Lo scopo è quello di notificare l'eventuale interruzione della trasmissione di uno dei due stream, fatto che ovviamente vanifica la procedura di sincronizzazione. Analogamente a *frame_start*, anche questo segnale viene inviato allo *Stream_aligner* per controllare lo stato di sincronizzazione dei due stream.

Per quanto riguarda i segnali interni:

- *End_count:* questo segnale funge da reset per il conteggio una volta raggiunta la dimensione del frame. In questo caso di utilizzo, dovendo trattare frame di 640x480 pixel, si è posto uguale al valore esadecimale 0x4AFFF. Di conseguenza, il segnale di conteggio è a 19 bit.
- *Count_value:* questo è il segnale di conteggio a 19 bit aggiornato all'interno del blocco always@ e resettato come in Fig.6.
- *En_count:* è il segnale che funge da enable del conteggio. Il suo aggiornamento avviene nello stato *Count* della FSM ed è subordinato al controllo sul segnale *tvalid*.
- *Flag:* questo segnale viene utilizzato per tenere traccia dell'avvenuto inizio di un frame e realizzare il comportamento del segnale *control*. L'aggiornamento di questo segnale avviene all'interno dello stato *frame_sync* nella FSM del contatore.

Il funzionamento della FSM integrata nel contatore verrà illustrato procedendo per ciascuno stato:

- *Idle:* questo è lo stato raggiunto al reset del contatore. L'uscita da questo stato è possibile solo quando il segnale *tvalid* è uguale a 1 e verso lo stato *count*.
- *Count:* analogamente allo stato precedente, il comportamento della macchina a stati viene deciso in base al valore di *tvalid*. Nel caso sia uguale a 1 viene abilitato il conteggio e lo stato futuro viene deciso in base al raggiungimento o meno del valore di fine conteggio. In caso contrario, lo stato futuro diventa nuovamente quello di *idle.*
- *Frame_sync:* questo stato viene raggiunto solo quando il conteggio raggiunge la dimensione del frame. Dunque, in questa situazione, viene rilevato l'inizio di un nuovo frame. I segnali *flag* e *frame_start* vengono settati a 1, mentre lo stato futuro dipende sempre dal valore di *tvalid.* In caso sia uguale a 1, lo stato futuro diventa *count*, altrimenti diventa *idle*.
- *Default:* lo stato futuro diventa quello di idle, mentre *flag* e *frame_start* sono settati a 0.

Il ruolo del modulo *Pixel_counter* è quindi quello di rilevare l'inizio di un nuovo frame trasmesso con l'AXI stream e di comunicarne l'eventuale interruzione allo *Stream_aligner*.

Stream Aligner

Il modulo *Stream_aligner* è una macchina a stati che prende in ingresso i segnali del protocollo AXI relativi ai due stream, i segnali di controllo generati dai due pixel counter e controlla la FIFO attraverso cui viene fatto passare lo stream contenente il frame di riferimento. È stato adottato questo approccio per evitare di bloccare lo stream proveniente dalla camera, essendo quello su cui vengono effettuate tutte le operazioni a monte. Il codice di tale modulo è il seguente:

```
module stream_aligner_new_fsm (
    input wire          clk,
    input wire          resetn,
    input wire          frame_start_new,
    input wire          frame_start_ref,
    input wire          control_new,
    input wire          control_ref,
    input wire          enable,
    input wire          tvalid_new,
    input wire          tvalid_ref,
    input wire          treadyOut_new,
    input wire          treadyOut_ref,
    output reg          tready_new,
    output reg          tready_ref,
    output reg          tvalidOut_new,
    output reg          tvalidOut_ref,
    output reg          sync_rst
    );

parameter [1:0] idle=2'd0, wait_for_sync=2'd1, synchronized=2'd2;
reg [1:0] cs;
reg [1:0] ns;

// Sequential FSM
always@(posedge clk, negedge resetn)
begin
    if (resetn==1'b0)
        cs<=idle;
    else
        cs<=ns;
end
```

Figure 8: Stream_aligner.v pt.1

```
// Combinational FSM
always@(*)
begin
    sync_rst=1'b0;

    case (cs)

        /* In the idle state the stream from the camera is let through regardless of the enable signal.
        Then, the enable signal is checked and triggers the synchronization procedure when equal to 1. */

        idle:
        begin
            tready_new = treadyOut_new;
            tready_ref = treadyOut_ref;
            tvalidOut_new = tvalid_new;
            tvalidOut_ref = tvalid_ref;
            if (enable==1'b0)
                ns=cs;
            else
            begin
                if (frame_start_ref==1'b1)
                begin
                    if (frame_start_new==1'b1)
                     begin
                        ns=synchronized;
                     end
                    else
                        ns=wait_for_sync;
                end
                else
                    ns=cs;
            end
        end
```

Figure 9: Stream_aligner.v pt.2

```
/* This state is reached when the SOF from the reference occurs --> the reference stream is stopped and we wait for the
    SOF from the camera to occur. */

wait_for_sync:
begin
    tready_new = treadyOut_new; // The stream from the camera is let through.
    tvalidOut_new = 1'b1;   // Output is valid.
    tvalidOut_ref = 1'b0;   // Output is invalid.
    if (frame_start_new==1'b1)
    begin
        ns=synchronized;    // The camera SOF has arrived --> the two streams can now be let through.
        tready_ref = treadyOut_ref;
        sync_rst = 1'b1;
    end
    else if(enable==1'b0)
    begin
        ns=idle;
        tready_ref=treadyOut_ref;
    end
    else
    begin
        ns=cs;              // We are still waiting for the reference SOF to arrive.
        tready_ref = 1'b0;   // Stop the reference stream and wait for the camera SOF to arrive.
    end
end
```

Figure 10: Stream_aligner.v pt.3

```
        /* This state is reached when the two streams are synchronized --> they are transmitted until one of them stops.
        Then, they need to be synchronized again. */

        synchronized:
        begin
            tready_new = treadyOut_new;
            tready_ref = treadyOut_ref;
            tvalidOut_new = tvalid_new;
            tvalidOut_ref = tvalid_ref;
            if (control_new==1'b0 || control_ref==1'b0)
                ns=idle;
            else if(enable==1'b0)
                ns=idle;
            else
                ns=cs;
        end

        default:
        begin
            ns=idle;
            tready_new = treadyOut_new;
            tready_ref = treadyOut_ref;
            tvalidOut_new = tvalid_new;
            tvalidOut_ref = tvalid_ref;
        end

    endcase
end

endmodule
```

Figure 11: Stream_aligner.v pt.4

Innanzitutto, verrà analizzata l'interfaccia dello *Stream_aligner*:

- *Enable:* questo segnale è lo stesso che abilita l'elaborazione all'interno del *Background_subtractor*. Il suo utilizzo nello stream aligner è fondamentale per decidere se sincronizzare gli stream o meno. Tale operazione, infatti, si rende necessaria solo nel caso sia richiesta l'elaborazione del frame acquisito dalla camera rispetto a quello di riferimento immagazzinato in memoria.
- *Frame_start_new/ref:* questi sono i segnali di frame_start generati dai due pixel counter, riferiti rispettivamente allo stream proveniente dalla camera e a quello di riferimento proveniente dal VDMA.
- *Control_new/ref:* analogamente ai frame_start, anch'essi sono generati dai due pixel counter.
- *Tvalid_new/ref:* sono i segnali del protocollo AXI stream. *Tvalid_new* proviene dal filtro di convoluzione a monte, mentre *Tvalid_ref* proviene da una delle FIFO di uscita del VDMA, in particolare quella posta in cascata al *ddr_to_axis_reader* che legge il frame di riferimento.
- *Tready_out_new/ref:* per questi segnali vale un discorso analogo a quello fatto per *Tvalid_new/ref*.
- *Tready_new/ref:* questi segnali sono generati dallo stream aligner e inviati al background subtractor. Il loro utilizzo è fondamentale al fine di bloccare la lettura dello stream di riferimento proveniente dal VDMA e poter quindi sincronizzare i due stream in ingresso al background subtractor.
- *Sync_rst:* questo segnale funge da reset per i pixel counter quando i due stream sono sincronizzati. In questo modo il conteggio procede allo stesso modo per i due AXI stream.

La sincronizzazione dei due stream è effettuata dalla seguente macchina a stati finiti:

- *Idle:* in questo stato lo stream aligner lascia passare entrambi gli stream senza effettuare alcuna operazione di sincronizzazione. Per decidere lo stato futuro viene effettuato un controllo sul valore del segnale di enable. Nel caso sia uguale a 0, non è necessario effettuare alcuna operazione di sincronizzazione, dunque lo stato futuro resta uguale a idle. In caso contrario invece si rende necessaria la sincronizzazione degli stream. Dunque, viene controllato il valore di frame_start_ref e, nel caso sia uguale a 1, viene controllato anche il valore di frame_start_new. Se entrambi risultano uguali a 1, i due stream sono già automaticamente sincronizzati e lo stato futuro diventa uguale a *synchronized*. Se invece frame_start_new è uguale a 0, lo stato futuro diventa uguale a *wait_for_sync.* Infine, nel caso *frame_start_ref* sia invece uguale a 0, lo stato futuro resta quello di idle.
- *Wait_for_sync:* in questo stato viene lasciato passare lo stream proveniente dal filtro di convoluzione e il relativo segnale di tvalid viene asserito a 1. Lo scopo di questo stato è aspettare che il segnale di frame_start dello stream proveniente dal filtro di convoluzione vada a 1, in modo tale da passare allo stato in cui i due stream sono sincronizzati. Un altro caso possibile è quello in cui il segnale di enable diventa uguale a 0: non dovendo più effettuare alcuna elaborazione tra i due stream, lo stato futuro diventa quello di idle ed entrambi gli stream sono lasciati passare. Infine, se nessuna delle condizioni precedenti risulta verificata, lo stato futuro resta uguale a quello presente e lo stream di riferimento proveniente dal VDMA viene bloccato.
- *Synchronized:* in questo stato gli stream sono sincronizzati, dunque vengono lasciati passare entrambi. Gli unici controlli effettuati in questo stato riguardano i segnali control generati dai pixel counter e quello di enable. Il primo controllo si rende necessario poiché l'interruzione di uno degli stream porta inevitabilmente alla perdita di sincronia e alla necessità di ripetere il procedimento, mentre lo switch dell'enable a 0 rende superflua la sincronizzazione degli stream.
- *Default:* analogamente al pixel counter, è presente uno stato di default in cui i segnali di uscita della FSM sono settati a valori di default per evitare situazioni non definite.

Ora saranno illustrate le modifiche apportate al progetto originale su Vivado SDK, al fine di implementare il meccanismo di acquisizione del frame di riferimento.

6: Modifiche su Vivado SDK

Le modifiche software apportate su SDK sono volte alla programmazione del nuovo modulo *ddr_to_axis_reader* inserito all'interno del VDMA e alla gestione dell'interrupt per l'acquisizione del frame di riferimento in memoria (tramite pressione del pulsante P16). Innanzitutto, per illustrare il procedimento seguito è utile mostrare il block design del VDMA modificato:

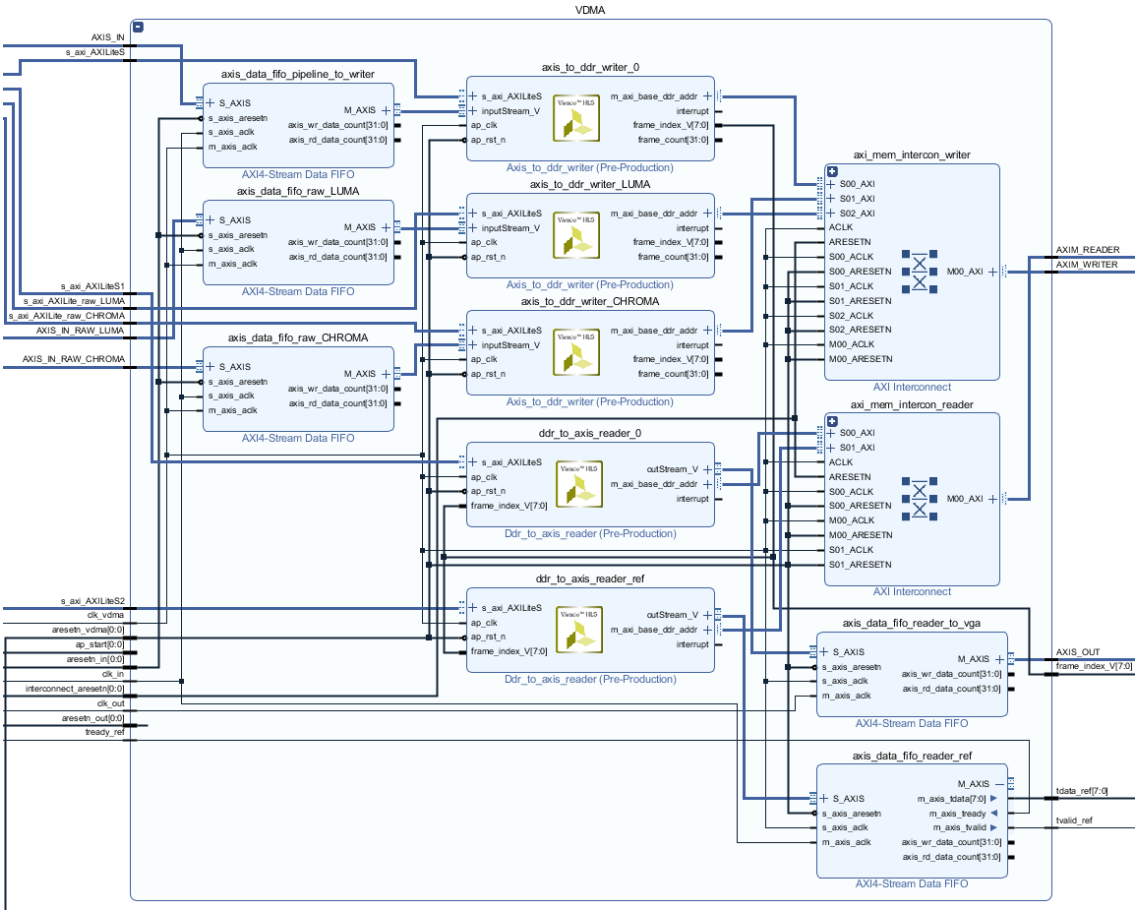

Figure 12: VDMA modificato

Di seguito è indicato il mapping in memoria delle diverse componenti immagine scritte e lette tramite il VDMA:

	Componente grayscale	Componente Chroma	Componente Luma	Frame di riferimento (grayscale)
Indirizzo iniziale	0x10000000	0x11000000	0x12000000	14000000
Indirizzo finale	0x112BFFFF	0x122BFFFF	0x132BFFFF	152BFFFF

La scelta dell'indirizzo iniziale per il frame di riferimento è stata presa in accordo con i limiti della memoria ddr indicati da Vivado SDK all'interno del file *lscript.ld*.

Di seguito è illustrato il funzionamento del VDMA modificato:

- *Ddr_to_axis_reader_0:* questo modulo effettua la lettura del frame memorizzato all'indirizzo FRAME_BUFFER_BASE_ADDR. Durante il funzionamento del sistema, a questo indirizzo verrà immagazzinato il frame attualmente acquisito dalla camera se l'enable è uguale a 0, oppure l'immagine elaborata se l'enable è uguale a 1.
- *Ddr_to_axis_reader_ref:* questo modulo effettua la lettura del frame memorizzato all'indirizzo FRAME_BUFFER_BASE_ADDR_REF. Questo è l'indirizzo al quale viene memorizzato il frame di riferimento all'arrivo dell'interrupt.
- *Axis_to_ddr_writer_0:* questo modulo effettua la scrittura di un frame in memoria ddr a partire da un indirizzo specificato in fase di configurazione. Essendo l'unico writer utilizzato per la componente grayscale, l'indirizzo di scrittura cambia a seconda dell'arrivo dell'interrupt o meno:
    - No interrupt: il writer scrive in memoria a partire dall'indirizzo FRAME_BUFFER_BASE_ADDR.
    - Arrivo dell'interrupt: il writer scrive in memoria a partire dall'indirizzo FRAME_BUFFER_BASE_ADDR_REF.

In questo modo, utilizzando un solo writer, è possibile effettuare lo storage in memoria di due frame distinti in due locazioni di memoria differenti. Di questi due frame, quello letto a partire da FRAME_BUFFER_BASE_ADDR verrà inviato al modulo AXIS_TO_VGA, mentre quello letto a partire da FRAME_BUFFER_BASE_ADDR_REF verrà retroazionato e posto in ingresso alla gerarchia *background_eraser*.

Questa soluzione è stata preferita rispetto alla creazione di un percorso parallelo all'interno del VDMA, approccio che avrebbe previsto anche l'inserimento di un secondo writer dedicato alla scrittura in memoria a partire da FRAME_BUFFER_BASE_ADDR_REF.

Di seguito sono illustrate le modifiche apportate ai file sorgente di SDK per configurare il nuovo reader e implementare l'interrupt connesso al pulsante P16.

Reader del frame di riferimento

L'aggiunta di un nuovo modulo *ddr_to_axis_reader* ha comportato le seguenti modifiche:

- Aggiunta di una funzione *init_ddr_to_axis_reader_ref* per l'inizializzazione del nuovo modulo.
- Aggiunta di una funzione *configue_ddr_to_axis_reader_ref* per la configurazione del nuovo modulo. In particolare, questo reader viene configurato con indirizzo di base uguale a FRAME_BUFFER_ BASE_ADDR_REF (0x14000000).

Ovviamente, al fine di garantirne il corretto funzionamento, è necessario aggiungere le chiamate a queste funzioni rispettivamente nei file *platform.c* e *application.c*.

Interrupt

L'approccio adottato richiede la configurazione di un nuovo interrupt e un'appropriata ISR per configurare correttamente il modulo writer. Dunque, oltre ad aggiungere le relative funzioni di enable, disable e configurazione al file *interrupts.c,* è stata scritta la seguente interrupt service routine:

```
int configure_capture_ref_interrupt(XScuGic* gicInst)
{
    if (XScuGic_Connect(gicInst, XPAR_FABRIC_CAPTURE_REF_INTR, capture_ref_handler, NULL) != XST_SUCCESS)
    {
        xil_printf("XScuGIc_Connect capture_ref failed!\n");
        return XST_FAILURE;
    }

    XScuGic_SetPriorityTriggerType(gicInst, XPAR_FABRIC_CAPTURE_REF_INTR, 0x10, 0x3); // Check priority
    printf("[INFO] Setting a lower priority (0x10) and rising edge sensitivity (0x3) to reset handler (SW1)\n");

    XScuGic_Disable(gicInst, XPAR_FABRIC_CAPTURE_REF_INTR);

    return XST_SUCCESS;
}

XGpio write_control_gpio;

void capture_ref_handler(void *ptr)
{
    int result;

    //XAxis_to_ddr_writer_DisableAutoRestart(&writer);
    result = configue_axis_to_ddr_writer_ref();
    if(result != XST_SUCCESS)
    {
        xil_printf("There is an error about configue_axis_to_ddr_writer_ref \n\r");
    }
    xil_printf("configue_axis_to_ddr_writer_ref done \n\r");

    usleep(SLEEP_TIME);

    XAxis_to_ddr_writer_DisableAutoRestart(&writer);
    result = configue_axis_to_ddr_writer();
    if(result != XST_SUCCESS)
    {
        xil_printf("There is an error about configue_axis_to_ddr_writer\n\r");
    }
    xil_printf("configue_axis_to_ddr_writer done \n\r");
}
```

Figure 13: Codice della ISR per l'acquisizione del frame di riferimento

Lo SLEEP_TIME è stato calcolato in modo tale da garantire la scrittura degli 8 frame buffer con cui è configurato l'axis_to_ddr_writer:

$$SLEEP\ TIME = \frac{640 * 480 * 8}{24 * 10^6\ Hz} = 0.1024\ s$$

Ricordando che i frame elaborati sono composti da 640x480 pixel, ciascuno codificato con 8 bit.

7: Risultati Sperimentali

Ora saranno mostrati i risultati del funzionamento del sistema per diversi valori della soglia di confronto, definita anch'essa come un valore a 8 bit. In seguito a diversi test, è stato individuato 30 come valore ottimale della soglia di confronto tra i due frame. Di seguito sono mostrati i risultati ottenuti per tre valori diversi della soglia.

Figura 14: Immagini acquisite dalla camera senza elaborazione

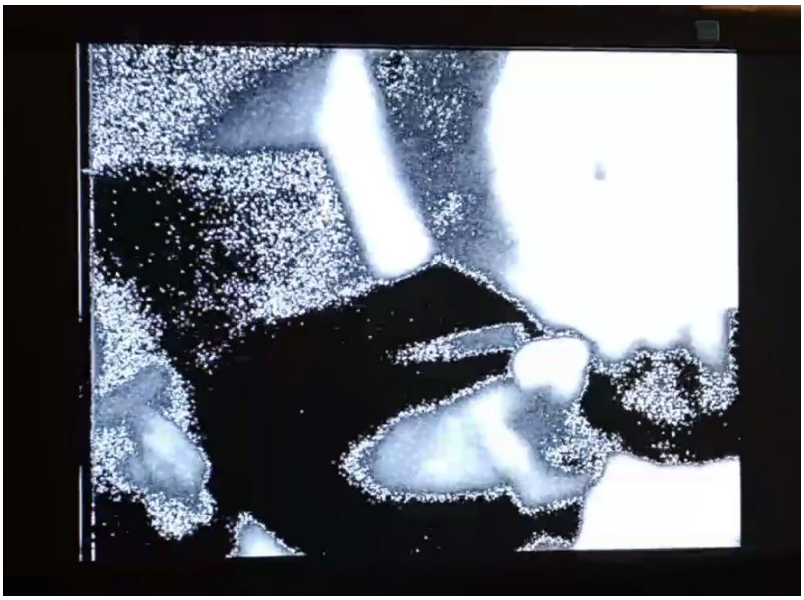

Figura 15: Immagine elaborata con soglia=10

Figura 16: Immagine elaborata con soglia=30 (solo background)

Figura 17: Immagine elaborata con soglia=30

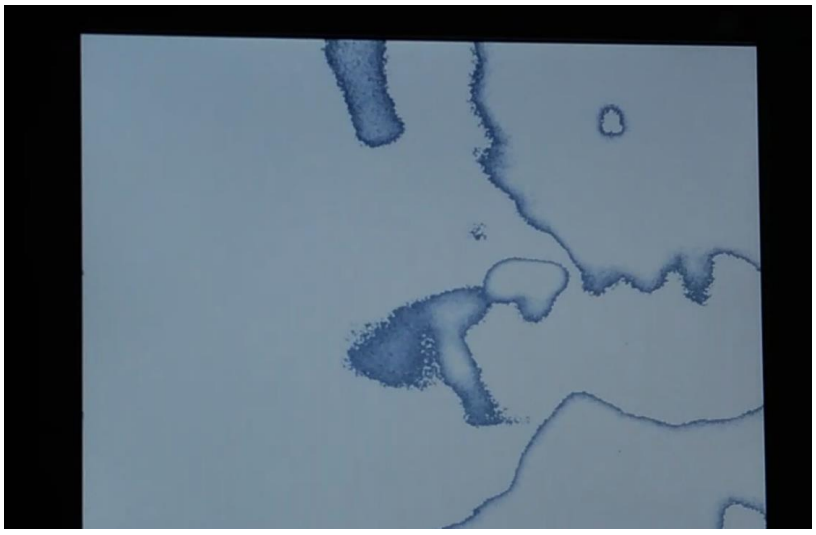

Figura 18: Immagine elaborata con soglia=70

In questo caso l'immagine di Fig.14 (in alto) è quella assunta come riferimento per l'elaborazione. È evidente come il rumore costituisca un disturbo notevole per l'elaborazione dell'immagine. Questo è particolarmente visibile con valori di soglia molto bassi (=10): ciò rientra nel comportamento atteso dato che anche piccole differenze di valore tra i pixel dovute al rumore portano al superamento della soglia di confronto.

Come era lecito aspettarsi, l'aumento della soglia (=70) porta ad una rimozione più uniforme dello sfondo, ma la definizione del soggetto inquadrato diminuisce rispetto al caso con soglia=30. Dunque, valori intermedi della soglia risultano essere quelli ottimali, ma vanno calibrati in relazione alla rumorosità dell'immagine acquisita.

Dunque, aumentare la soglia del confronto può essere utile in situazioni particolarmente rumorose, dato che in questo modo l'effetto del rumore viene attenuato a scapito della definizione del soggetto inquadrato.

Dall'osservazione in tempo reale del monitor, si nota che alcuni frame non vengono elaborati correttamente. Questo può essere imputato alla scrittura in memoria del frame di riferimento poiché per scrivere gli 8 frame buffer è stata utilizzata la funzione *usleep,* che tuttavia non garantisce che l'ultimo frame buffer sia scritto per intero.

8: Conclusioni e Sviluppi Futuri

Si può quindi concludere che il sistema progettato effettua con successo la sincronizzazione di due stream video e lo storage di due frame in locazioni diverse della memoria. È tuttavia evidente una certa sensibilità al rumore introdotto dal sensore, anche per valori intermedi della soglia di confronto.

Sviluppi futuri potrebbero essere:

- Riduzione della rumorosità dell'immagine acquisita tramite configurazione più accurata del sensore o attraverso il filtro di convoluzione.
- L'utilizzo di un algoritmo più sofisticato e preciso per la rilevazione delle differenze tra un frame e l'altro.
- Implementazione di una soglia di confronto programmabile dal processore.
- L'utilizzo di un interrupt per segnalare al processore la conclusione della scrittura degli 8 frame buffer di riferimento garantendo che tutti i frame buffer scritti in memoria siano completi.