**“High Level Synthesis of Digital Systems”**

**2023-2024**

**Prof. PERRI**

**Prof. FRUSTACI**

**FIR Filter Analysis**

|  |  |
| --- | --- |
| **Date** | <24/04/2024> |
| **Document** | Final Document |

|  |  |  |
| --- | --- | --- |
| **Full Name** | **ID** | **E-mail Address** |
| Giorgio Ubbriaco | 247284 | bbrgrg00h11d086x@studenti.unical.it |

Contents

[Definitions 4](#_Toc167188159)

[I. Tasks to be performed 5](#_Toc167188160)

[II. Abstract 6](#_Toc167188161)

[1. Introduction 7](#_Toc167188162)

[2. Solutions 8](#_Toc167188163)

[2.1. Code Hoisting Solution 8](#_Toc167188164)

[2.2. Loop Fission Solution 8](#_Toc167188165)

[2.3. Loop Unrolling Solution Factor=2 and Factor=4 9](#_Toc167188166)

[2.3.1. Manual Unrolling Solution 9](#_Toc167188167)

[2.3.2. Automatic Unrolling Solution 9](#_Toc167188168)

[2.3.3. Automatic Unrolling and Partitioning Solution 9](#_Toc167188169)

[2.4. Operation Chaining Solution 9](#_Toc167188170)

[2.5. Loop Pipelining Solution 9](#_Toc167188171)

[2.6. Bitwidth Optimization Solution 9](#_Toc167188172)

[2.7. AXI Solution 9](#_Toc167188173)

# Definitions

# Tasks to be performed

# Abstract

# Introduction

# Solutions

Qui di seguito verranno illustrate e analizzate diverse implementazioni del filtro FIR. In particolare, verranno mostrati i report generati da HLS e Vivado così da effettuare ulteriori considerazioni a riguardo.

## Code Hoisting Solution

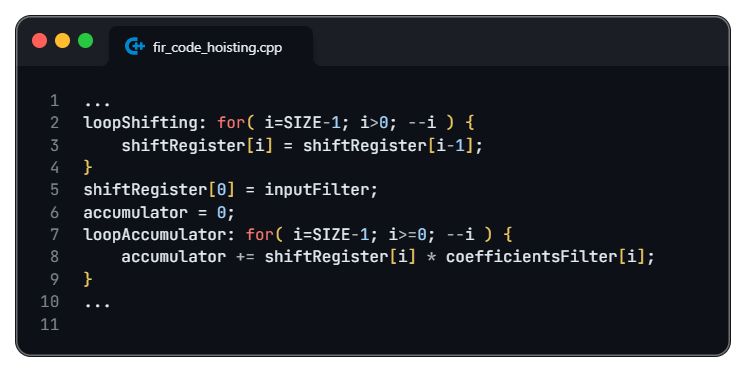
Dal momento che l’implementazione software del filtro FIR prevede un’istruzione condizionale, , dal punto di vista hardware questo corrisponderebbe ad un overhead a livello di risorse. Questo è dovuto al fatto che si dovrebbe leggere il valore dell’indice nella struttura dati che lo contiene e prevedere un comparatore così da confrontare tale valore con il valore e, pertanto, effettuare l’operazione logica di tra i bit che rappresentano la variabile e quelli rappresentanti il valore .



Quello che dovremmo aspettarci è che ci sia una riduzione dell’utilizzazione delle risorse prevista dal tool di HLS e dal tool di Vivado e, inoltre, una riduzione di un’unità del trip count rispetto alla soluzione non ottimizzata dovuto al fatto che il caso viene gestito al di fuori del ciclo for comportando, pertanto, un’iterazione in meno.

## Loop Fission Solution

La soluzione in questione prevede lo splitting del ciclo , previsto nell’implementazione software, in due parti. In particolare, l’operazione di shifting verrà effettuata in un ciclo denominato , mentre l’operazione di accumulo in un ciclo denominato . Questa divisione del ciclo in due cicli differenti permette al tool di HLS di effettuare ottimizzazioni indipendenti su entrambe le operazioni poiché presenti in un due cicli distinti. Bisogna notare però che, in questo caso, avrò due cicli e, pertanto, lo scheduling delle operazioni sarà differente dagli altri.



Quello che dovremmo aspettarci è che il trip count relativo al primo ciclo for sia pari a 10 poiché il caso viene gestito al di fuori del ciclo nella successiva istruzione similmente come avviene nella soluzione del code hoisiting, mentre il trip count relativo al secondo ciclo ci aspettiamo sia pari a 11.

## Loop Unrolling Solution Factor=2 and Factor=4

### Manual Unrolling Solution

### Automatic Unrolling Solution

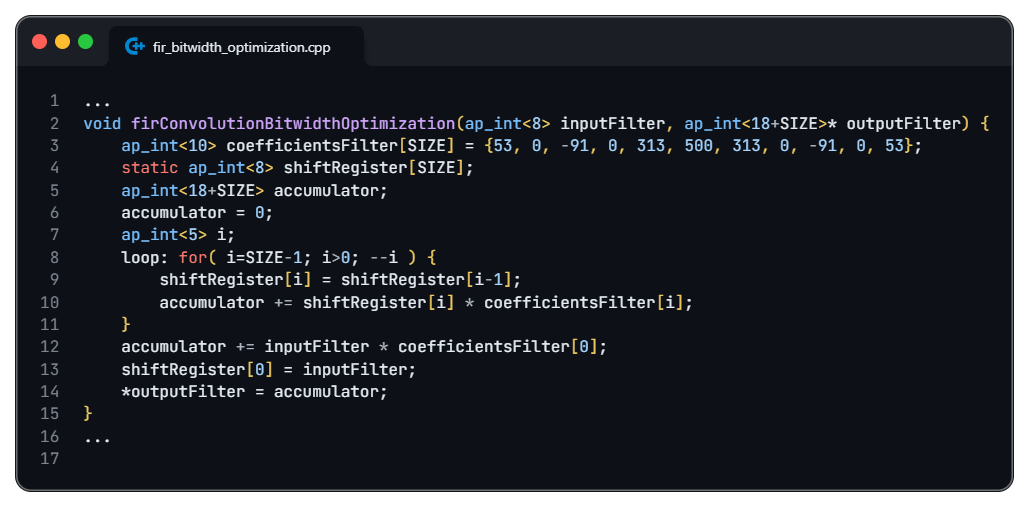
### Automatic Unrolling and Partitioning Solution

## Operation Chaining Solution

## Loop Pipelining Solution

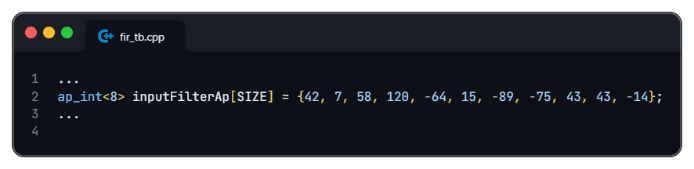
## Bitwidth Optimization Solution

Nelle varie soluzioni precedentemente citate, il tipo di dato principalmente utilizzato è stato il tipo , avente una precisione pari a 32 bit. Considerando che non tutte le variabili hanno bisogno di questa quantità di bit come, ad esempio, la variabile indice del ciclo for, il tipo per i coefficienti già noti, è possibile predisporre di una precisione arbitraria. In particolare, utilizzando la libreria è possibile definire le variabili con un numero di bit ben preciso. Ad esempio, sta ad indicare che l’array di lunghezza possiede al suo interno valori aventi ognuno una rappresentazione basata su 10 bit. In questo caso, questo è possibile poiché i coefficienti sono già noti a priori e, pertanto, è possibile prevedere la quantità di bit per la loro rappresentazione. Quindi, facendo un ragionamento analogo per l’input, l’output e le variabili interne al metodo, descritto a livello software, è possibile definire arbitrariamente il numero di bit per la loro rappresentazione. In questo modo, sarà possibile riscontrare un abbattimento dell’utilizzazione delle risorse a livello hardware.



In particolare, l’input prevede un numero di bit pari a 8 poiché è stato scelto un input noto a priori. Nel caso in cui si volesse prevedere un input di tipologia bisognerebbe fare un ragionamento analogo ma considerando un numero di bit pari a 32.

Considerando il ciclo , in particolare l’operazione di moltiplicazione comporta un risultato pari alla somma dei bit degli input corrispondenti e, pertanto, un numero di bit pari a 10+8, poiché rispettivamente 10 è il numero di bit dei coefficienti considerati per il filtraggio mentre 8 è il numero di bit considerati per l’input noto.



In particolare, scegliendo come input i valori sopra mostrati e considerando il valore massimo presente in tale array che li contiene, cioè il valore 120, è stata scelta una precisione pari a 8 bit poiché tramite 8 bit e considerando una variabile con segno è possibile rappresentare il range di valori . Analogamente per la precisione relativa ai coefficienti, considerando il valore massimo, cioè 500, e che si tratta di valori con segno, è stata scelta una precisione pari a 10 bit tramite la quale è possibile rappresentare valori compresi nel range .

Pertanto, dal momento che l’array shiftRegister contiene i valori presenti all’interno dell’array inputFilter, anch’esso conterrà valori aventi precisione ognuno a 8 bit. Invece, per quanto riguarda la variabile accumulator, essa presenterà una precisione pari a 10+8+SIZE, dove è pari a 11 e rappresenta la capacità dell’array di shifting, cioè il numero di elementi che può contenere. La dimensione per tale variabile è stata scelta in questo modo poiché al suo interno, per ogni iterazione del ciclo , viene registrato un valore derivante da una moltiplicazione e una successiva addizione. In particolare, la precisione viene aumentata di un fattore poiché per ogni iterazione del ciclo, la precisione della variabile aumenta dal momento che una somma presenta come risultato un valore avente una precisione pari a quella dell’input con precisione maggiore aumentata di un’unità.

Infine, per la variabile indice , dal momento che deve iterare nel range di valori , è stata scelta una precisione pari 5 bit poiché considerando che si tratta di una variabile con segno, è in grado di ricoprire il range di valori . Ovviamente l’iterazione è stata esclusa poiché le istruzioni corrispondenti sono state considerate al di fuori del ciclo .

## AXI Solution