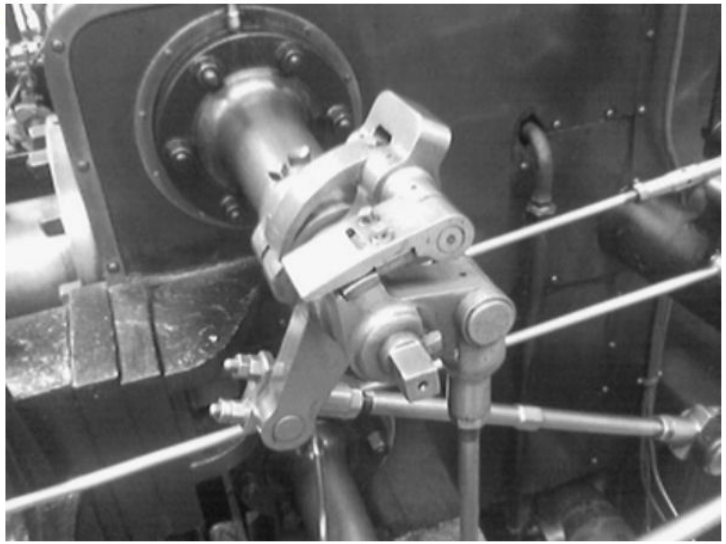
Filtro 1d usando un core HLS con interfacce AXI-streaming

Introduzione: filtro 1d (immagine)

In [12]:
```
from PIL import Image
rgb = Image.open("pictures/valve.png")
bw = rgb.convert("L")
bw = bw.resize((400,300),Image.ANTIALIAS)
bw
```

Out[12]:



Introduzione: filtro 1d (immagine filtrata)

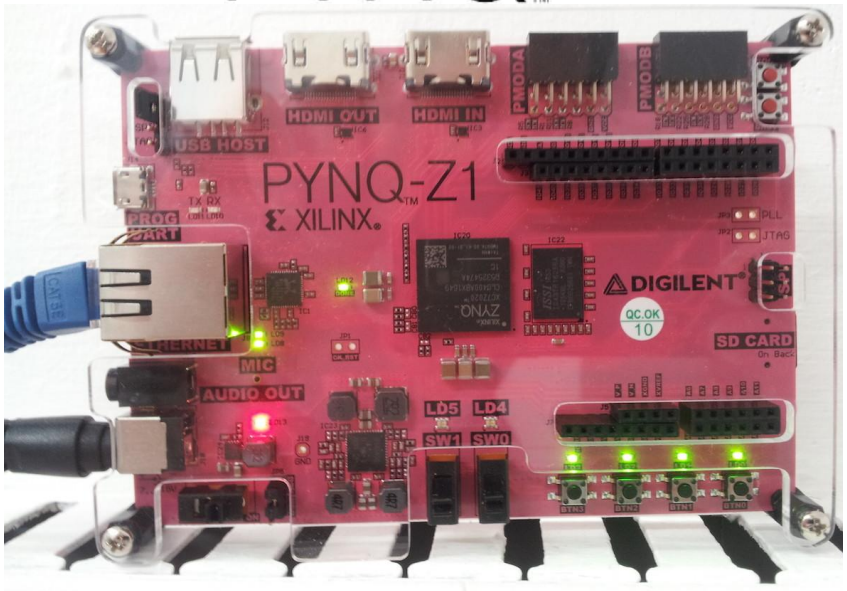
In [13]:
```
coeffs = [0, 0, -1, -2, 0, 2, 1, 0, 0]
sig = list(bw.getdata())
output = overlay.run(coeffs, sig)

out = Image.new(bw.mode, bw.size)
out.putdata(output)
out
```

Out[13]:

XILINX- PYNQ: PYTHON PRODUCTIVITY FOR ZYNQ + Vivado HLS

PYNQ™



Schema di realizzazione

PC

1. Realizzazione di un core HLS:
   a. Codice C/C++-like compilato da HLS e che produce un file RTL (register-transfer level, astrazione usata nei linguaggi di descrizione hardware, HDL, per creare rappresentazioni di alto livello di un circuito)
2. Creazione di un Bitstream FPGA:
   a. Inserimento dell'IP core realizzato all'interno di un progetto per FPGA
   b. Wiring
   c. Produzione del bitstream

PYNQ

3. PYTHON
   a. Utilizzazione del bitstream e del core HLS attraverso uno script in python sulla PYNQ

3. PYTHON

1. __init__.py
   a. File di python che definisce un pacchetto importabile di python
2. stream.py
   a. Definisce la classe streamOverlay di python che interagisce con il bitstream dell'FPGA
   b.

```
1 from pynq import Overlay, GPIO, Register
2 from pynq import Xlnk
3 import numpy as np
4 import os
5 import inspect
```

3. PYTHON: stream.py

```
29      def __init__(self, bitfile, **kwargs):
30          file_path = os.path.abspath(inspect.getfile(inspect.currentframe()))
31          dir_path = os.path.dirname(file_path)
32          bitfile = os.path.join(dir_path, bitfile)
33          super().__init__(bitfile, **kwargs)
34          self.__resetPin = GPIO(GPIO.get_gpio_pin(0), "out")
35          self.__ap_ctrl = Register(self.filt1d.mmio.base_addr, 32)
36          self.xlnk = Xlnk()
```

Carica il bit file

Definisce manualmente il pin di reset del GPIO

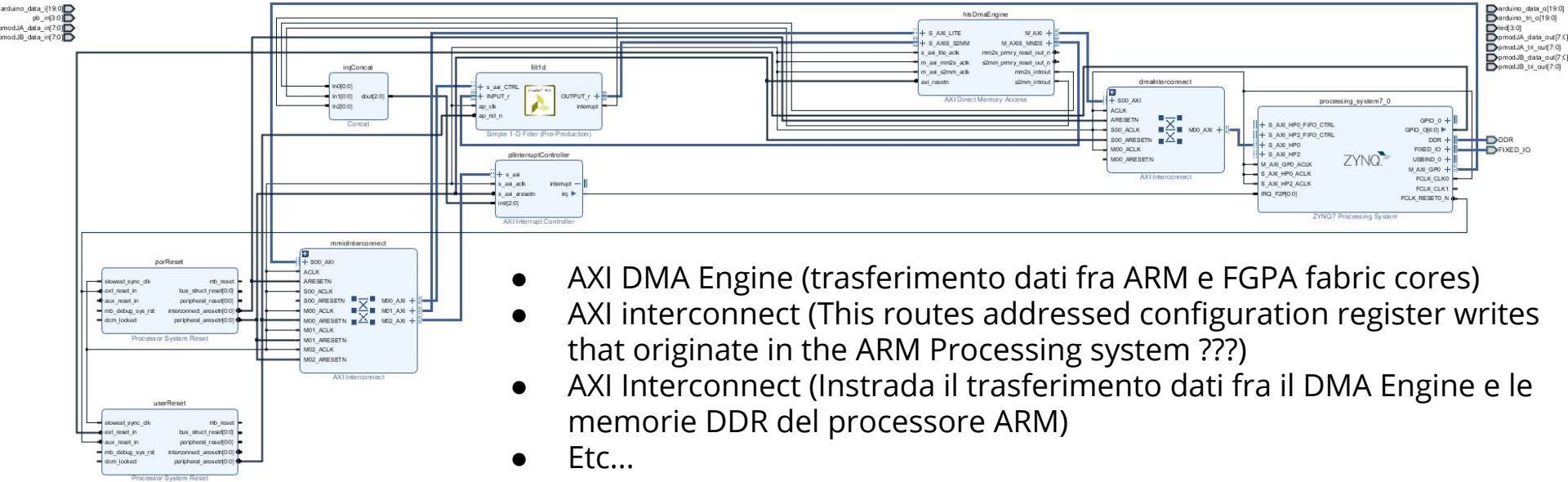
Definisce un oggetto registro all'indirizzo 0x0 negli indirizzi di filt1d

Definisce il "contiguous memory manager"

3. PYTHON: stream.py

```
66      def run(self, coeffs, buf):
67  Setup   # -------------------- Part 1: --------------------
68          self.nreset()                                          Toglie il reset, permettendo ai registri di poter essere letti e scritti nel core HLS
69          l = len(buf)
70          self.__load(coeffs, l)                                 Carica i coefficienti e la lunghezza nei registri del core
71          cmabuf_src = self.xlnk.cma_array([l], np.int32)        Crea un array di dati di input contigui per il DMA Engine
72          for i in range(l):
73              cmabuf_src[i] = buf[i]
74          cmabuf_dest = self.xlnk.cma_array([l], np.int32)       Crea un array di dati di output contigui per il DMA Engine
75  Execution # -------------------- Part 2: --------------------
76          self.hlsDmaEngine.recvchannel.transfer(cmabuf_dest)    Transizione dati FPGA → ARM nel DMA Engine sul canale di ricezione
77          self.hlsDmaEngine.sendchannel.transfer(cmabuf_src)     Transizione dati ARM → FPGA nel DMA Engine sul canale di trasmissione
78          self.__start()
79          self.hlsDmaEngine.sendchannel.wait()
80          self.hlsDmaEngine.recvchannel.wait()
81          self.__stop()
82  Cleanup # -------------------- Part 3: --------------------
83          buf = cmabuf_dest.tolist()                             Prende il segnale filtrato come una lista python dal buffer di output del DMA
84          cmabuf_dest.freebuffer()
85          cmabuf_src.freebuffer()
86          return buf
```

2. Creazione di un Bitstream FPGA



- AXI DMA Engine (trasferimento dati fra ARM e FGPA fabric cores)
- AXI interconnect (This routes addressed configuration register writes that originate in the ARM Processing system ???)
- AXI Interconnect (Instrada il trasferimento dati fra il DMA Engine e le memorie DDR del processore ARM)
- Etc...

- Output
  - stream.tcl (Tool Command Language)
  - stream.bit (bitstream file)

} Trasferiti sulla PYNQ

1. Realizzazione di un core HLS

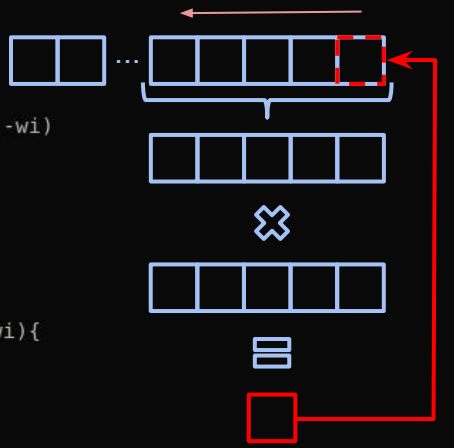
- filt1d.cpp
  - Fa un filtro unidimensionale su un array in input e lo mette in un array di output
  - Il filtro è un prodotto scalare fra un array di coefficienti e un intervallo di valori in input
  - Utilizzo di pragma-instruction per dare le opportune istruzioni ad HLS

```
1  void filt1d(axis_t *INPUT, axis_t *OUTPUT,int coeff[C_NUM_COEFF], unsigned int length)
2  {
3  #pragma HLS INTERFACE axis depth=50 port=INPUT
4  #pragma HLS INTERFACE axis depth=50 port=OUTPUT
5  #pragma HLS INTERFACE s_axilite port=coeff  bundle=CTRL
6  #pragma HLS INTERFACE s_axilite port=length bundle=CTRL
7  #pragma HLS INTERFACE s_axilite port=return bundle=CTRL
8  #pragma HLS ARRAY_PARTITION COMPLETE variable=coeff
9    int window[C_NUM_COEFF] = {0};
10 #pragma HLS ARRAY_PARTITION COMPLETE variable=window
11   int sum;
12   axis_t cur;
13   for (unsigned int i = 0 ; i < length; i++)
14   {
15 #pragma HLS PIPELINE
16     for (unsigned int wi = C_NUM_COEFF-1; wi > 0; --wi)
17     {
18 #pragma HLS UNROLL
19       window[wi] = window[wi - 1];
20     }
21     cur = *INPUT++;
22     window[0] = cur.data;
23     sum = 0;
24     for (unsigned int wi = 0; wi < C_NUM_COEFF; ++wi){
25 #pragma HLS UNROLL
26       sum += coeff[wi] * window[wi];
27     }
28     cur.data =sum;
29     *OUTPUT++ = cur;
30   }
31 }
32
```

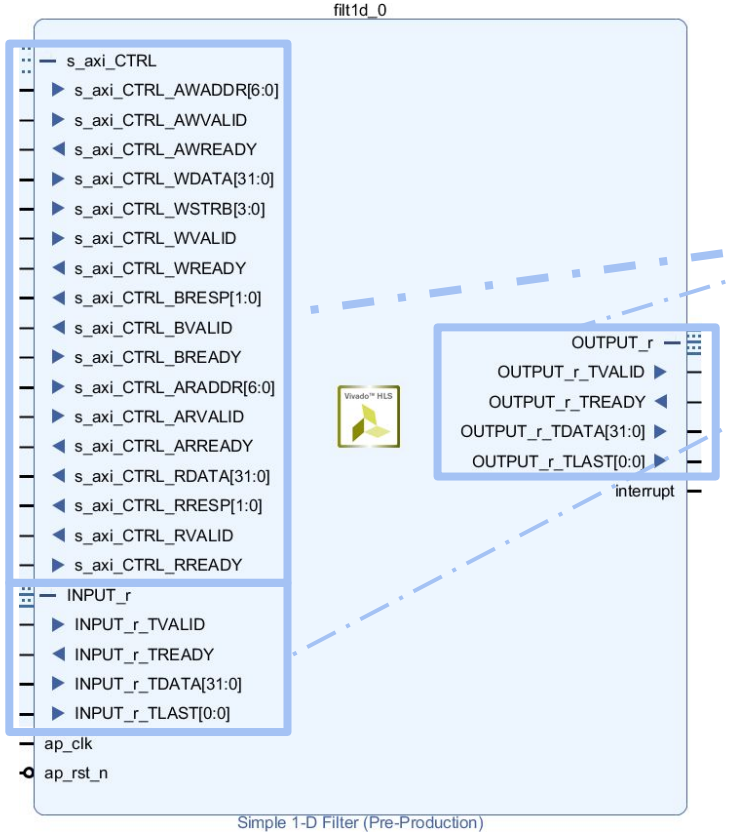
Porte di input e output come interfacce AXI-stream

AXI-Lite bus chiamato CTRL

Array di registri hardware



1. Realizzazione di un core HLS




```
1  void filt1d(axis_t *INPUT, axis_t *OUTPUT,int coeff[C_NUM_COEFF], unsigned int length)
2  {
3  #pragma HLS INTERFACE axis depth=50 port=INPUT
4  #pragma HLS INTERFACE axis depth=50 port=OUTPUT
5  #pragma HLS INTERFACE s_axilite port=coeff   bundle=CTRL
6  #pragma HLS INTERFACE s_axilite port=length bundle=CTRL
7  #pragma HLS INTERFACE s_axilite port=return bundle=CTRL
8  #pragma HLS ARRAY_PARTITION COMPLETE variable=coeff
9    int window[C_NUM_COEFF] = {0};
10 #pragma HLS ARRAY_PARTITION COMPLETE variable=window
11   int sum;
12   axis_t cur;
13   for (unsigned int i = 0 ; i < length; i++)
14   {
15 #pragma HLS PIPELINE
16     for (unsigned int wi = C_NUM_COEFF-1; wi > 0; --wi)
17     {
18 #pragma HLS UNROLL
19       window[wi] = window[wi - 1];
20     }
21     cur = *INPUT++;
22     window[0] = cur.data;
23     sum = 0;
24     for (unsigned int wi = 0; wi < C_NUM_COEFF; ++wi){
25 #pragma HLS UNROLL
26       sum += coeff[wi] * window[wi];
27     }
28     cur.data =sum;
29     *OUTPUT++ = cur;
30   }
31 }
32
```

Porte di input e output come interfacce AXI-stream

AXI-Lite bus chiamato CTRL

Array di registri hardware