Fast FPGA prototyping: an application of Fast Gradient Method to audio signal processing

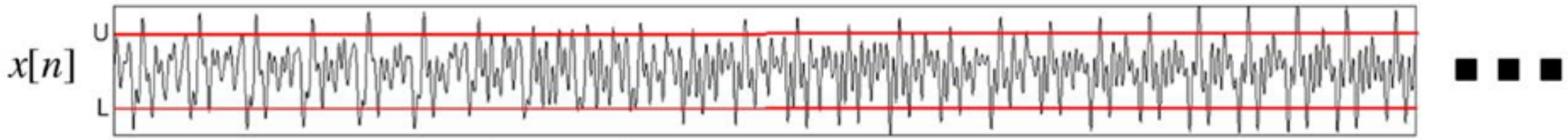
Andrea Suardi
IMPERIAL COLLEGE LONDON
a.suardi@imperial.ac.uk

Outline

- Case study: real-time clipping audio signal with FGM
- Automated circuit design:
  - C coding
  - Validation (Software simulation)
  - Build FPGA circuit (IP)
- IP verification: Hardware In the Loop framework

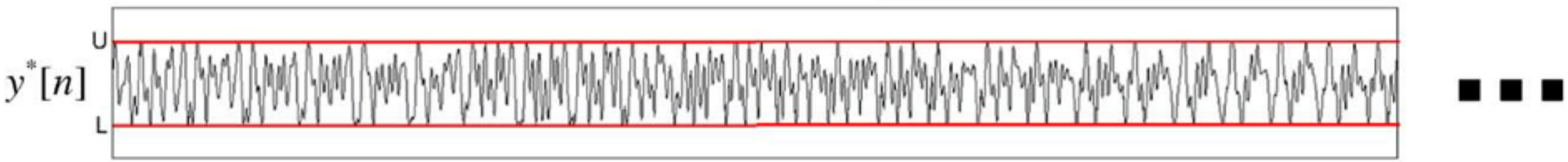
Real-time clipping of audio signal with FGM

Bruno Defraene



Fast Gradient Method

Configuration parameters



Fast Gradient for audio processing

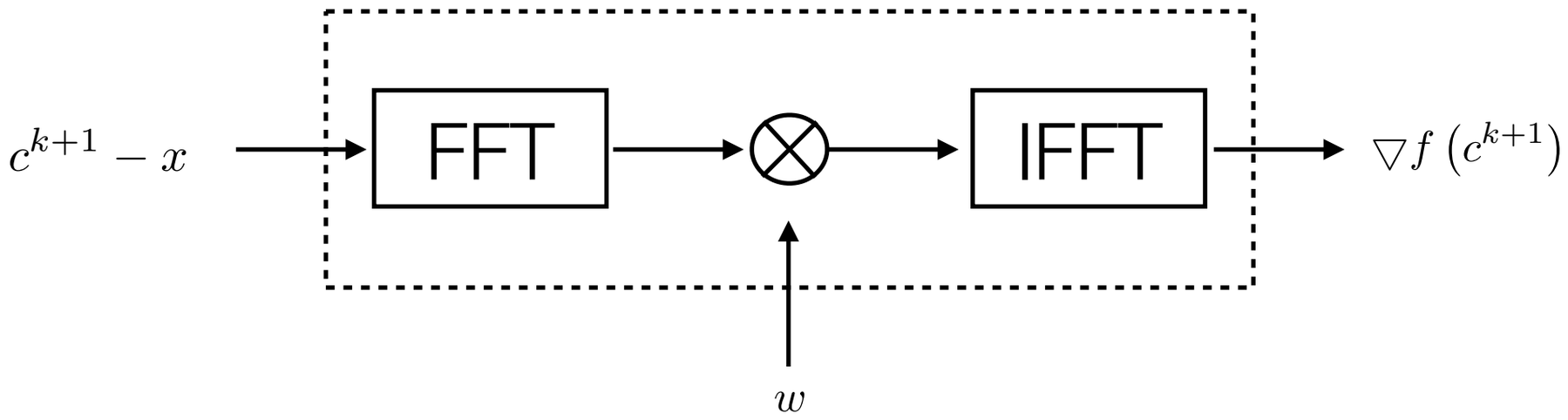
Input $\mathbf{x} \in \mathbb{R}^N$, $\mathbf{w} \in \mathbb{R}^N$, $K_{\max}$, $\boldsymbol{\delta} = [\delta^0 \delta^1 \dots \delta^{K_{\max}-1}]^T \in \mathbb{R}^{K_{\max}}$ L, U, C^{-1}
Output $\mathbf{y}^* \in \mathbb{R}^N$

$\mathbf{y}^0 = \mathbf{c}^0 = \mathbf{x}$
$\nabla f(\mathbf{c}^0) = \mathbf{0}$
$k = 0$
while $k < K_{\max}$ **do**
  $\tilde{\mathbf{y}}^{k+1} = \mathbf{c}^k - C^{-1} \nabla f(\mathbf{c}^k)$
  $\mathbf{y}^{k+1} = \Pi_Q(\tilde{\mathbf{y}}^{k+1})$
  $\mathbf{c}^{k+1} = \mathbf{y}^{k+1} + \delta^k (\mathbf{y}^{k+1} - \mathbf{y}^k)$
  $\nabla f(\mathbf{c}^{k+1}) = \mathbf{D}^H \mathrm{diag}(\mathbf{w}) \mathbf{D}(\mathbf{c}^{k+1} - \mathbf{x})$
  $k = k + 1$
end while
$\mathbf{y}^* = \mathbf{y}^k$

Fast Gradient for audio processing

Input $\mathbf{x} \in \mathbb{R}^N$, $\mathbf{w} \in \mathbb{R}^N$, $K_{\max}$, $\boldsymbol{\delta} = [\delta^0 \delta^1 \dots \delta^{K_{\max}-1}]^T \in \mathbb{R}^{K_{\max}}$ L, U, C^{-1}
Output $\mathbf{y}^* \in \mathbb{R}^N$

$\mathbf{y}^0 = \mathbf{c}^0 = \mathbf{x}$
$\nabla f(\mathbf{c}^0) = \mathbf{0}$
$k = 0$
while $k < K_{\max}$ **do**
  $\tilde{\mathbf{y}}^{k+1} = \mathbf{c}^k - C^{-1} \nabla f(\mathbf{c}^k)$
  $\mathbf{y}^{k+1} = \Pi_Q(\tilde{\mathbf{y}}^{k+1})$
  $\mathbf{c}^{k+1} = \mathbf{y}^{k+1} + \delta^k(\mathbf{y}^{k+1} - \mathbf{y}^k)$
  $\nabla f(\mathbf{c}^{k+1}) = \mathbf{D}^H \mathrm{diag}(\mathbf{w}) \mathbf{D}(\mathbf{c}^{k+1} - \mathbf{x})$
  $k = k + 1$
end while
$\mathbf{y}^* = \mathbf{y}^k$



Automated circuit design … C coding

Input $\mathbf{x} \in \mathbb{R}^N$, $\mathbf{w} \in \mathbb{R}^N$, $K_{\max}$, $\delta = [\delta^0 \delta^1 \ldots \delta$
Output $\mathbf{y}^* \in \mathbb{R}^N$

$\mathbf{y}^0 = \mathbf{c}^0 = \mathbf{x}$
$\nabla f(\mathbf{c}^0) = \mathbf{0}$
$k = 0$
while $k < K_{\max}$ **do**
  $\tilde{\mathbf{y}}^{k+1} = \mathbf{c}^k - C^{-1} \nabla f(\mathbf{c}^k)$
  $\mathbf{y}^{k+1} = \Pi_Q(\tilde{\mathbf{y}}^{k+1})$
  $\mathbf{c}^{k+1} = \mathbf{y}^{k+1} + \delta^k(\mathbf{y}^{k+1} - \mathbf{y}^k)$
  $\nabla f(\mathbf{c}^{k+1}) = \mathbf{D}^H \mathrm{diag}(\mathbf{w}) \mathbf{D}(\mathbf{c}^{k+1} - \mathbf{x})$
  $k = k + 1$
end while
$\mathbf{y}^* = \mathbf{y}^k$

```
#define Kmax 30
#define FRACTION_LENGTH 24

#define N 512

typedef ap_fixed< FRACTION_LENGTH+3,3,AP_TRN, AP_SAT> data_t;

void clip(
          data_t x[N],
          data_t w[N],
          data_t bmin[N],
          data_t bmax[N],
          data_t delta[Kmax],
          data_t lipschitz,
          data_t y_out[N])
{

//variables
data_t Grad[N];
data_t Grad_lipschitz[N];
data_t new_Grad[N];
data_t y_tilde[N];
data_t y_new[N];
data_t y[N];
data_t y_delta[N];
data_t y_delta_delta[N];
data_t c_new[N];
data_t c[N];

int k,i;
```

Automated circuit design … C coding

Input $\mathbf{x} \in \mathbb{R}^N$, $\mathbf{w} \in \mathbb{R}^N$, $K_{\max}$, $\boldsymbol{\delta} = [\delta^0 \delta^1 \dots \delta^{K_{\max}-1}]^T \in \mathbb{R}^{K_{\max}}$ L, U, C^{-1}
Output $\mathbf{y}^* \in \mathbb{R}^N$

1: $\mathbf{y}^0 = \mathbf{c}^0 = \mathbf{x}$
2: $\nabla f(\mathbf{c}^0) = 0$
3: $k = 0$
4: **while** $k < K_{\max}$ **do**
5: $\quad \tilde{\mathbf{y}}^{k+1} = \mathbf{c}^k - C^{-1} \nabla f(\mathbf{c}^k)$
6: $\quad \mathbf{y}^{k+1} = \Pi_Q(\tilde{\mathbf{y}}^{k+1})$
7: $\quad \mathbf{c}^{k+1} = \mathbf{y}^{k+1} + \delta^k(\mathbf{y}^{k+1} - \mathbf{y}^k)$
8: $\quad \nabla f(\mathbf{c}^{k+1}) = \mathbf{D}^H \mathrm{diag}(\mathbf{w}) \mathbf{D}(\mathbf{c}^{k+1} - \mathbf{x})$
9: $\quad k = k + 1$
10: **end while**
11: $\mathbf{y}^* = \mathbf{y}^k$

```
//initialization
initialization_loop: for (i=0; i< N; i++)
{
    Grad[i]=0;
    c[i]=x[i]
    y[i]=x[i];
}
```

Automated circuit design … (

Input $\mathbf{x} \in \mathbb{R}^N$, $\mathbf{w} \in \mathbb{R}^N$, $K_{\max}$, $\boldsymbol{\delta} = [\delta^0 \delta^1 \dots \delta^{K_{\max}-1}]^T \in \mathbb{R}^{K_{\max}}$ $L, U,$
Output $\mathbf{y}^* \in \mathbb{R}^N$

$\mathbf{y}^0 = \mathbf{c}^0 = \mathbf{x}$
$\nabla f(\mathbf{c}^0) = \mathbf{0}$
$k = 0$
while $k < K_{\max}$ **do**
  $\tilde{\mathbf{y}}^{k+1} = \mathbf{c}^k - C^{-1} \nabla f(\mathbf{c}^k)$
  $\mathbf{y}^{k+1} = \Pi_Q(\tilde{\mathbf{y}}^{k+1})$
  $\mathbf{c}^{k+1} = \mathbf{y}^{k+1} + \delta^k(\mathbf{y}^{k+1} - \mathbf{y}^k)$
  $\nabla f(\mathbf{c}^{k+1}) = \mathbf{D}^H \mathrm{diag}(\mathbf{w}) \mathbf{D}(\mathbf{c}^{k+1} - \mathbf{x})$
  $k = k + 1$
end while
$\mathbf{y}^* = \mathbf{y}^k$

```
// Fast Gradient iterations loop
FG_loop:for (int k=0; k< Kmax; k++)

    //Iteration
    inner_loop_row:    for(i = 0; i < N; i++)
    {
        //Gradient * Lipschitz
        Grad_lipschitz[i] = Grad[i] * lipschitz;

        //unconstrained update
        y_tilde[i]=c[i]-Grad_lipschitz[i];

        //projection
        if (y_tilde[i]>bmax[i])
            y_new[i]=bmax[i];
        else if (y_tilde[i]<bmin[i])
            y_new[i]=bmin[i];
        else
            y_new[i]=y_tilde[i];

        //update c
        y_delta[i]=y_new[i]-y[i];
        y_delta_delta[i]=delta[k] * y_delta[i];
        c_new[i]=y_new[i]+y_delta_delta[i];

        to_fft[i]=c_new[i]-x[i];
    }

    // FFT
    hls::fft(to_fft, fft_out);

    //apply weights
    w_loop: for (i=0; i< N; i++)
    {
        to_ifft[i].real()=fft_out[i].real()*w[i];
        to_ifft[i].imag()=fft_out[i].imag()*w[i];
    }

    // IFFT
    hls::ifft(to_ifft, new_Grad);

    //update variables
    update_loop: for (i=0; i< N; i++)
    {
        Grad[i]=new_Grad[i];
        c[i]=c_new[i]
        y[i]=y_new[i];
    }
}
```

Automated circuit design … C coding

Input $\mathbf{x} \in \mathbb{R}^N$, $\mathbf{w} \in \mathbb{R}^N$, $K_{\max}$, $\boldsymbol{\delta} = [\delta^0 \delta^1 \dots \delta^{K_{\max}-1}]^T \in \mathbb{R}^{K_{\max}}$ L, U, C^{-1}
Output $\mathbf{y}^* \in \mathbb{R}^N$

$\mathbf{y}^0 = \mathbf{c}^0 = \mathbf{x}$
$\nabla f(\mathbf{c}^0) = \mathbf{0}$
$k = 0$
while $k < K_{\max}$ **do**
  $\tilde{\mathbf{y}}^{k+1} = \mathbf{c}^k - C^{-1} \nabla f(\mathbf{c}^k)$
  $\mathbf{y}^{k+1} = \Pi_Q(\tilde{\mathbf{y}}^{k+1})$
  $\mathbf{c}^{k+1} = \mathbf{y}^{k+1} + \delta^k(\mathbf{y}^{k+1} - \mathbf{y}^k)$
  $\nabla f(\mathbf{c}^{k+1}) = \mathbf{D}^H \mathrm{diag}(\mathbf{w}) \mathbf{D}(\mathbf{c}^{k+1} - \mathbf{x})$
  $k = k + 1$
end while
$\mathbf{y}^* = \mathbf{y}^k$

```
//update output
update_output_loop: for (i=0; i< N; i++)
{
    y_out[i]=y[i];
}
```

Automated circuit design … C coding

1. Set **data representation**

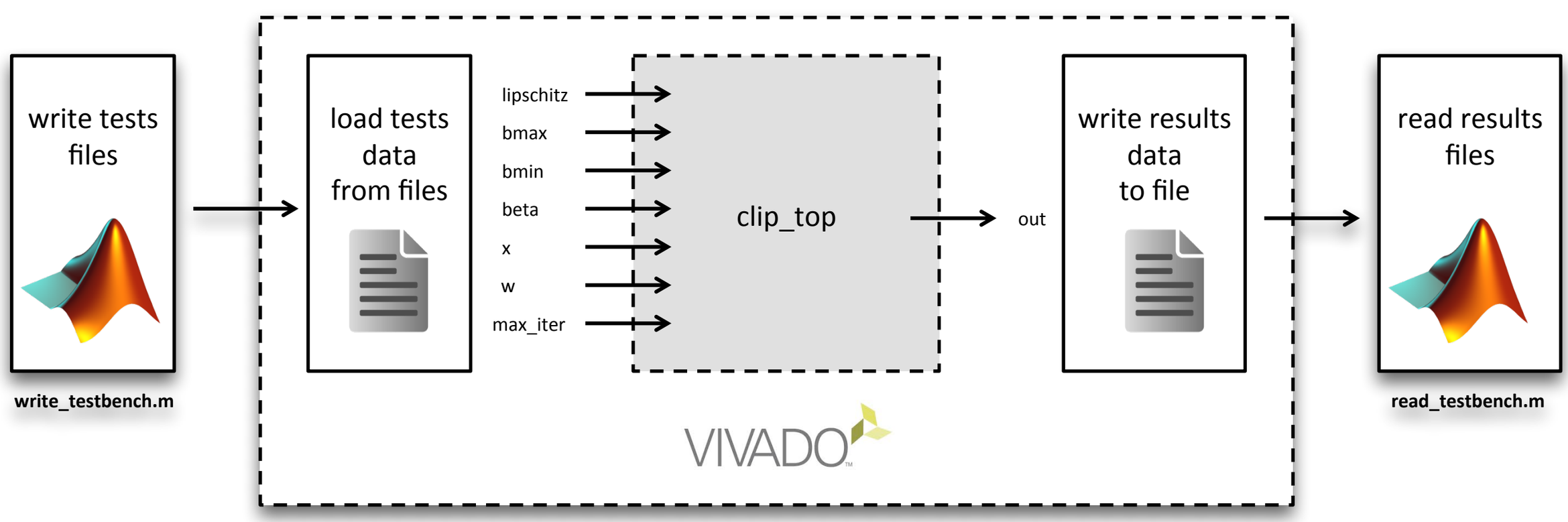
   - fixed-point (any)
   - floating point single precision

2. Set **number of iterations**

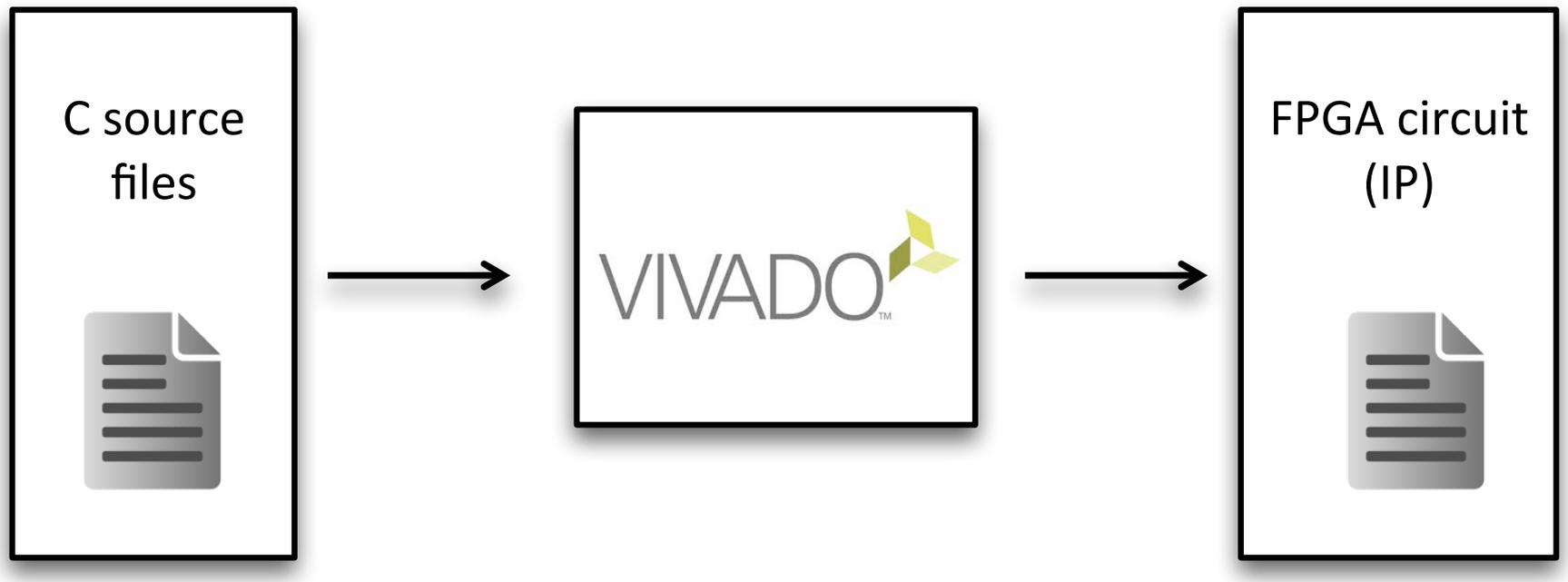
```
#define Kmax 30

#define FRACTION_LENGTH 24

typedef ap_fixed<FRACTION_LENGTH+3,3,AP_TRN, AP_SAT> data_t;
```

Automated circuit design … Software simulation (**Xilinx Vivado HLS)**

- **Simulate algorithm** behaviour with circuit bit accurate C model

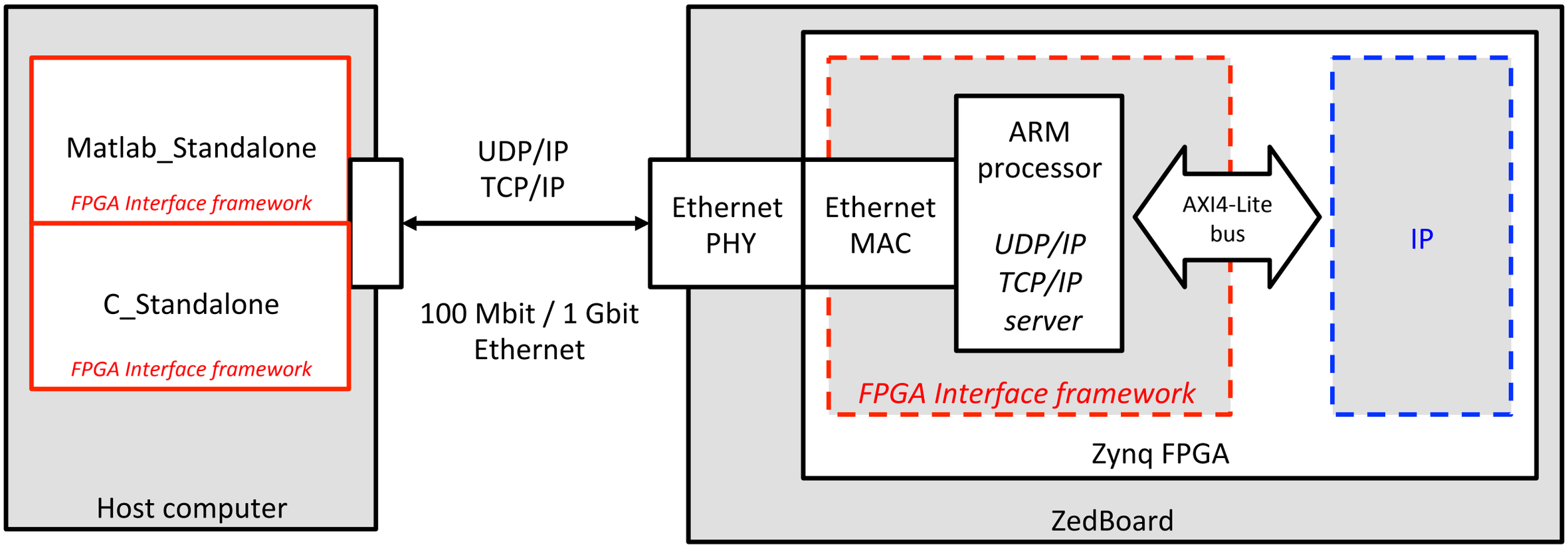


Automated circuit design … Build FPGA circuit (**Xilinx Vivado HLS)**



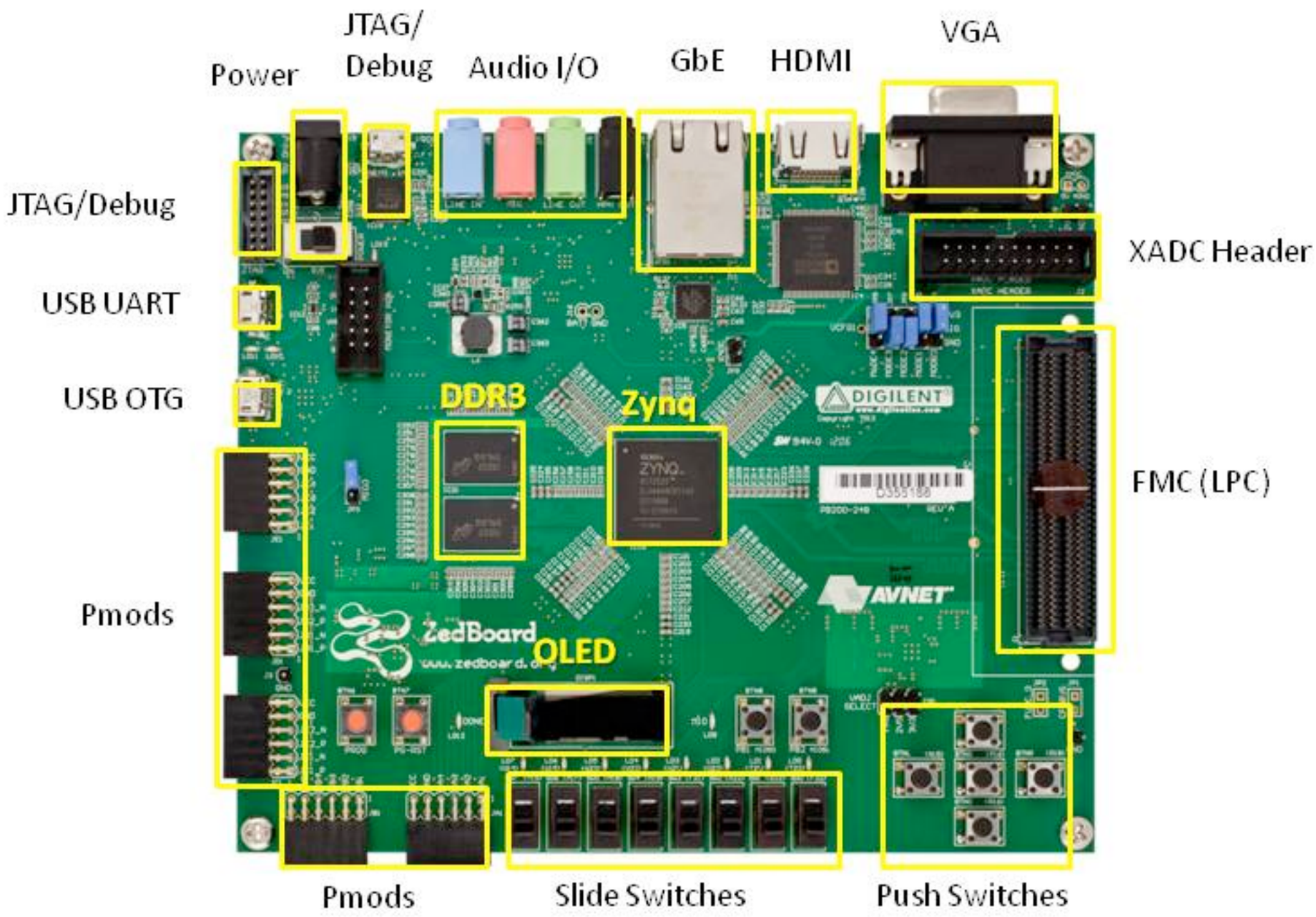
- Estimate resources (amount of silicon)
- Compute exact execution time

IP verification: Hardware In the Loop framework



FPGA Interface framework:

- C and Matlab library (Linux/Window)
- UDP/IP and TCP/IP ethernet interface
- Transfer data to/from the custom circuit (IP)
- Data format: 64/32 bits words, any buffer length
- Performance benchmark (execution time: FPGA and Host)



* SD card cage and QSPI Flash reside on backside of board

Step by step tutorials and source code will be available **open source** from **April 2014** on Imperial College Circuits and Systems website (http://www3.imperial.ac.uk/circuitssystems)

or email me (a.suardi@imperial.ac.uk)