!Shazam

1. Abstract:

!Shazam este un instrument digital, portabil și extrem de rapid, folosit pentru recunoașterea melodiilor, pentru ca utilizatorii să poată descoperi noi *jam*-uri cu precizie și ușurință. Pentru a oferi aceste caracteristici am decis să folosim puterea de procesare înaltă pe care FPGA-urile o permit datorită arhitecturii capabile să efectueze multe operații în paralel. Astfel, ne propunem să înregistrăm sunetele folosind placa audio a plăcii. După achiziția completă a datelor vom folosi o transformată Fourier rapidă care să aplice o convoluție (de tip periodic) pentru a găsi frecvențele undelor sinusoidale care ar putea alcătui semnalul. Dacă au fost folosite ***N*** intrări (sample-uri), se pot detecta maxim ***N/2*** frecvențe distincte care alcătuiesc semnalul (din cauza procesului de aliasing). Deorece urechea umană poate percepe unde sinusoide cu frecvențe între 20Hz și 20kHz, vom folosi un model care poate determina un maxim de $2^{14} = 16384$ frecvențe distincte în intervalul $[0, 16384]$, ceea ce înseamnă că vom avea nevoie de $2^{15} = 32768$ intrări pentru a avea o reprezentare care poate distinge sunetele suficient de similar cu urechea umană. Dacă presupunem o perioadă $\Delta T_{achiziționare}$ în care placa așteaptă toate intrările și o perioadă $\Delta T_{procesare}$ în care placa realizează calculele pentru determinarea frecvențelor (timp în care placa nu mai poate stoca alte input-uri) atunci numim $\Delta T_{SW} = \Delta T_{achiziționare} + \Delta T_{procesare}$ durata de timp a unui „sliding-window”. Putem atunci considera orice melodie ca alcătuită din P perioade $P = \frac{T_{melodie}}{\Delta T_{SW}}$. Pentru fiecare din aceste perioade se va alege semnalul sinusoidal cu cea mai mare energie stocată ($E = \frac{1}{2} \rho \cdot A \cdot f^2 \cdot \lambda$ - care depinde cel mai mult de frecvența f și amplitudinea A). Astfel putem realiza un algoritm care ține cont de unda cu cea mai mare energie în ΔT_{SW} și momentul de timp T în care a fost înregistrată. Această pereche energie-timp va fi stocată în baza de date. Un cântec va avea P perechi energie-timp stocate la ID-ul acestuia. Pentru algoritmul de *recunoaștere* vom compara cele X pachete venite cu cele P pachete stocate și vom realiza o histogramă cu numărul perioadelor relative care verifică o relație de adiacență când două *hash*-uri prezintă un *match*.

2. Implementare:

Vom folosi placa DE10-LITE deoarece reprezintă o variantă minimală care acoperă toate cerințele proiectului (intrare analog, chip ADC, unitate de procesare și pini pentru comunicarea cu o placă Arduino).

2.1. Achiziția datelor:

Pentru achiziția datelor vom lega un microfon la intrarea A0 a connector-ului Arduino, care va mapa în interior Canalul 0 al plăcii de sunet:

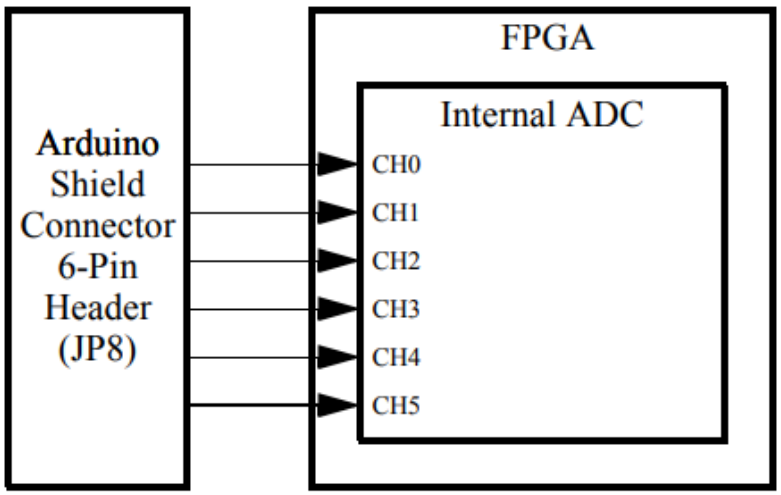


Figure 3. Signals to and from the ADC on the **DE10-Lite**

(Pentru un scurt tutorial care prezintă modul în care putem citi valorile analog de la CH-uri consultați pagina 59 din manualul plăcii: DE10-LITE Manual)
După specificația prezentată mai sus, cât și în Using the DE-Series ADC Controller (intel.com), vom primi sample-uri de 12 biți odată la 16 clock cycle-uri. Utilizând un calcul simplu, presupunând frecvența plăcii de 10 MHz $\Rightarrow \Delta T_{ACHIZ\ MIN} = 0.054s$ pentru 32.768 sample-uri.

2.2 Procesarea datelor

Pentru procesarea datelor vom utiliza specificațiile din articolul (PDF) The Fast Fourier Transform in Hardware: A Tutorial Based on an FPGA Implementation (researchgate.net) care presupune un *data-width* de 12 biți. Unitatea centrală de procesare a FFT-ului va fi modulul Butterfly care primește 32.768 de intrări și produce 32.768 ieșiri. Datorită algoritmului de generare al perechilor de *input-uri*:

Iteration level i ↓	Butterfly address pairs j ⟶
Level 0:	$\{0,1\}\ \{2,3\}\ \{4,5\}\ \{6,7\}\ \{8,9\}\ \cdots$
Level 1:	$\{0,2\}\ \{1,3\}\ \{4,6\}\ \{5,7\}\ \{8,10\}\ \cdots$
Level 2:	$\{0,4\}\ \{1,5\}\ \{2,6\}\ \{3,7\}\ \{8,12\}\ \cdots$
Level 3:	$\{0,8\}\ \{1,9\}\ \{2,10\}\ \{3,11\}\ \{4,12\}\ \cdots$
⋮	

putem instanția la fiecare nivel 32.768 de module care primesc ca input pentru cei doi operanzi:

```
module two_operand_butterfly_eval (
  input [11: 0] A,
  input [11: 0] B,
  input [11: 0] twiddle

  output [11: 0] out1,
  output [11: 0] out2,
);

  // pseudocode:
  assign out1 = A + B * twiddle;
  assign out2 = A - B * twiddle;
endmodule
```

unde:

```
// memory location for input A in decimal form
integer m;

wire [11: 0] A = prev_butterfly_output[m: m + 11];

// memory location for input B in decimal form
integer n;
wire [11: 0] B = prev_butterfly_output[n: n + 11];
```

iar rezultatul fiecărui modul va fi stocat într-o memorie RAM (aici numită *prev_butterfly_output*). Algoritmul este:

$$m = \textbf{Rotate}_5(2j, i)$$

$$n = \textbf{Rotate}_5(2j+1, i)$$

where $\textbf{Rotate}_N(x, y)$ indicates a circular left shift of N bit word x by y bits.

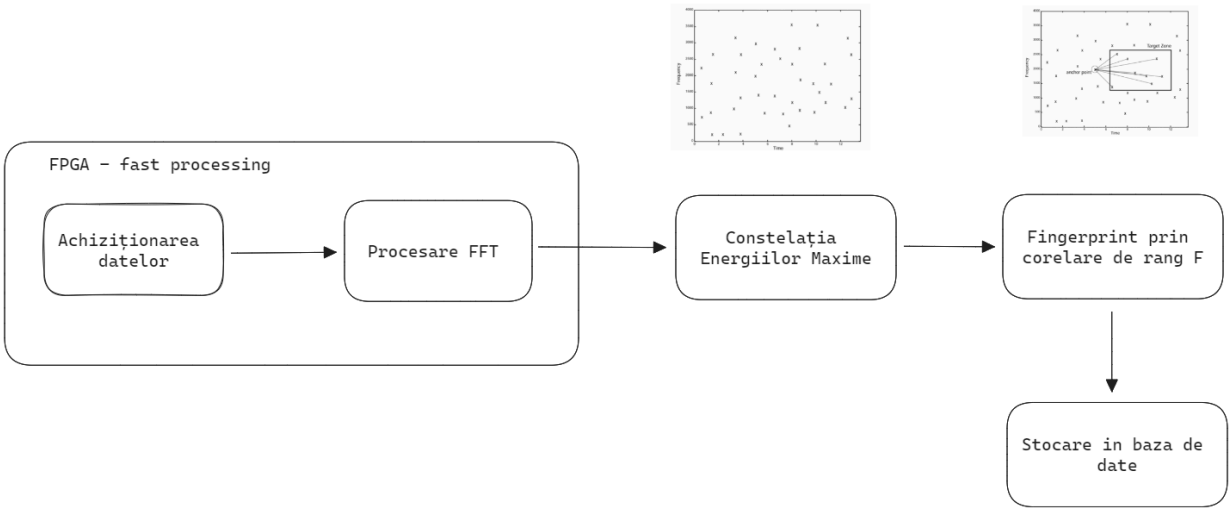
După cum se specifică și în articol, vom folosi operații cu virgulă fixă deoarece input-ul este de lungime fixă (12 biți).
Twiddle-urile vor fi urcate în memoria ROM a plăcii. Pentru a extrage adresa *twiddle-ul* de la nivelul i pentru perechea j, putem folosi:

The twiddle factor addresses are found by masking out the $N-1-i$ least significant bits of j.

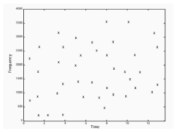
După ce toate ultimul nivel a fost computat, încărcăm datele pe Arduino, unde va avea loc algoritmul de *fingerprint*.

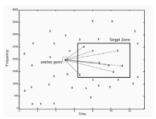
2.3. Diagrama de decizie:

Placa va avea trei stări principale: „Waiting”, „Training” și „Recognizing”:
Dacă este selectat modul „Training”:



Dacă este selectat modul „Recognizing”:





FPGA - fast processing

Achiziționarea datelor

Procesare FFT

Constelația Energiilor Maxime

Fingerprint prin corelare de rang F

Căutare folosind histograme de corelări multiple