Sisteme Incorporate

Cursul 14: Exemplu de implementare a unui sistem embedded

Cuprins

- Cum se poate proiecta o camera foto digitala simpla
- Perspectiva proiectantului
- Specificatii
- Design
  - Patru tipuri de implementare

Introducere

- Asamblarea puzzle-ului
  - Procesor general-purpose
  - Procesor single-purpose
    - Specializat
    - Standard
  - Memorie
  - Interfete
- Care sunt principalele probleme in proiectarea unei camere digitale
  - Alegerea unui procesor general-purpose vs. single-purpose
  - Partitionarea functionalitatii intre diferitele tipuri de procesoare

Care sunt functiile de baza ale unei camere digitale?

- Capteaza imagini
- Stocheaza imaginile in format digital
  - Fara film
  - Mai multe imagini stocate pe aceeasi camera
    - Numarul lor este d.p. cu capacitatea memoriei si numarul de octeti ocupati de o imagine
- Descarca imaginile pe un PC
- Aparute relativ recent pe piata
  - Systems-on-a-chip
    - Mai multe memorii si procesoare pe acelasi chip
  - Memorie flash de capacitate mare
- Exemplul de aici descrie o camera foarte simpla
  - O camera reala are multe alte functii pe langa acestea
    - Imagini de dimensiune diferita, managementul imaginilor pe card, procesare on-board, zoom etc.

Din perspectiva proiectantului

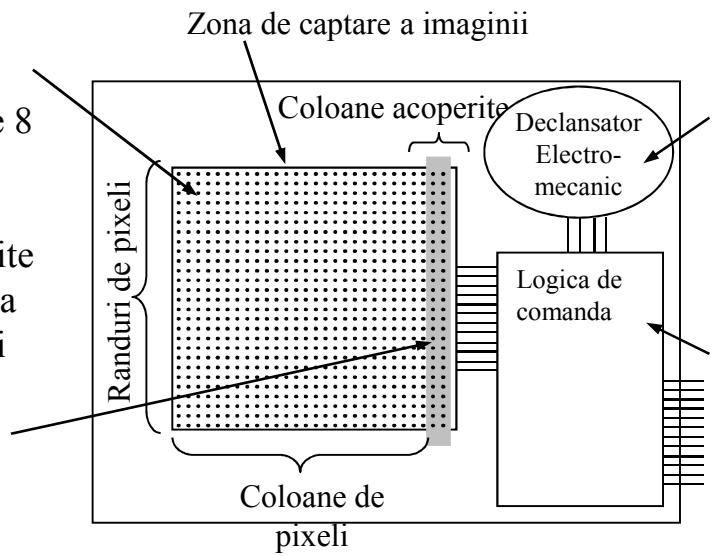
- Doua procese principale
  - Procesarea imaginii si stocarea in memorie
    - Cand este apasat declansatorul:
      - Captura de imagine
      - Converteste imaginea de la CCD in reprezentare binara
      - Comprima si arhiveaza imaginea in memorie
  - Descarcarea imaginilor pe un PC
    - Camera este atasata la PC
    - Un soft special comanda camera sa transmita imaginile captate pe interfata de comunicatie (seriala, USB etc)

Charge-coupled device (CCD)

- Senzorul care este responsabil de captura imaginii
- Dispozititv semiconductor cu structura multi-celulara, sensibil la lumina

Fiecare celula capata o sarcina electrica atunci cand este expusa la lumina. Aceasta sarcina poate fi convertita intr-o valoare pe 8 biti (0=negru, 255 alb)

Unele coloane sunt acoperite cu vopsea neagra. Valoarea masurata de la acesti pixeli este folosita pentru corectarea deviatiei de la zero a celorlalte celule.

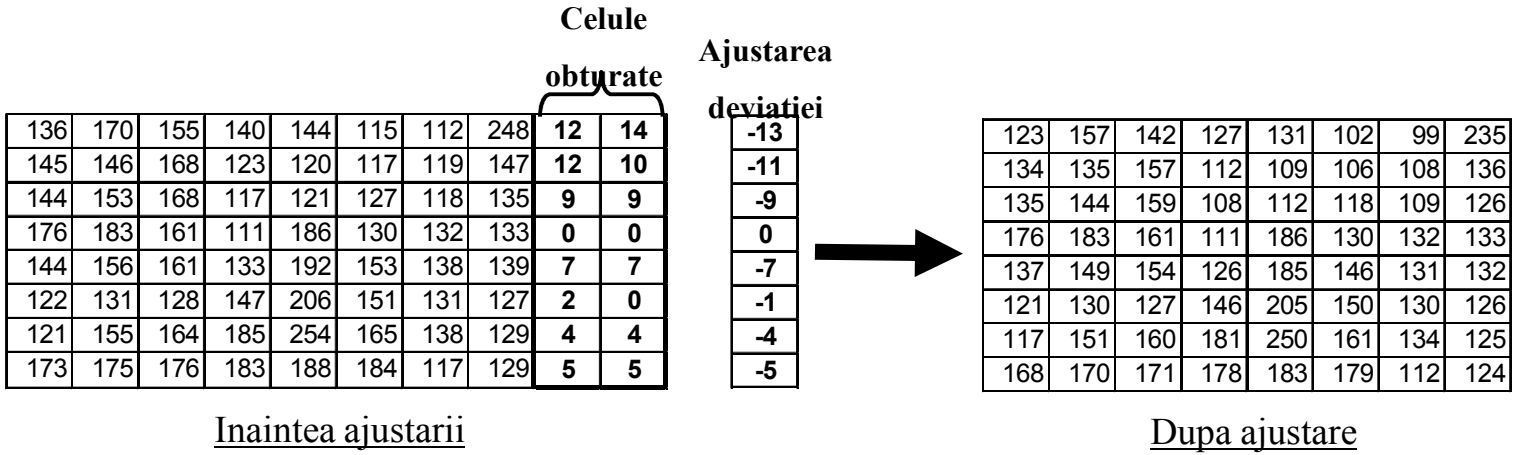


Declansatorul este actionat pentru a permite expunerea senzorului la lumina pentru o perioada controlata de timp

Logica de comanda, descarca celulele, activeaza declansatorul si citeste valorile de 8 biti lungime ale fiecarei celule. Valorile pot fi transmise urmatorului etaj de prelucrare printr-o interfata paralela de mare viteza.

Deviatia de la zero

- Din cauza erorilor de fabricatie, unele celule dau valori putin mai mari sau mai mici decat iluminarea adevarata.
- Eroarea este de obicei aceeasi pentru toate coloanele dar este diferita de la rand la rand
- O parte din celulele cel mai din stanga sunt obturate cu vopsea neagra
  - Daca acestea dau o valoare diferita de 0 -> eroarea de deviatie
  - Fiecare rand este corectat prin scaderea valorii medii pentru celulele obturate din randul respectiv



								Celule obturate	
136	170	155	140	144	115	112	248	12	14
145	146	168	123	120	117	119	147	12	10
144	153	168	117	121	127	118	135	9	9
176	183	161	111	186	130	132	133	0	0
144	156	161	133	192	153	138	139	7	7
122	131	128	147	206	151	131	127	2	0
121	155	164	185	254	165	138	129	4	4
173	175	176	183	188	184	117	129	5	5

Inaintea ajustarii

Ajustarea deviatiei
-13
-11
-9
0
-7
-1
-4
-5

123	157	142	127	131	102	99	235
134	135	157	112	109	106	108	136
135	144	159	108	112	118	109	126
176	183	161	111	186	130	132	133
137	149	154	126	185	146	131	132
121	130	127	146	205	150	130	126
117	151	160	181	250	161	134	125
168	170	171	178	183	179	112	124

Dupa ajustare

Compresie

- Avantaj: stocheaza mai multe imagini
- Transmiterea imaginilor la PC dureaza mult mai putin
- JPEG (Joint Photographic Experts Group)
  - Standardul cel mai folosit pentru compresia imaginilor
  - Pune la dispozitie mai multe moduri si optiuni de compresie
  - Algoritmul prezentat aici da cele mai bune rezultate de compresie. DCT (discrete cosine transform)
  - Imaginea este divizata in blocuri de 8 x 8 pixeli
  - Pentru fiecare bloc se executa trei pasi
    - DCT
    - Cuantificare
    - Codare Huffman

DCT

- Transforma matricea originala 8x8 din domeniul timp in domeniul frecventa
  - Valorile din contul stanga-sus contin informatii cantitative despre imagine
  - Coltul din dreapta-jos contine informatii despre detaliile fine
    - Se poate reduce cantitatea de informatie din acest colt fara a afecta prea mult calitatea imaginii percepute
- Transformarea directa FDCT (Forward DCT)
  - Folosita la compresia din imagine neprelucrata (RAW) in JPEG
- Transformata inversa IDCT (Inverse DCT)
  - Procesul invers prin care se obtin datele originale

Cuantificare

- Obtine rata mare de compresie prin reducerea cantitatii de informatie
  - Reduce numarul de biti cu care este codificata imaginea
    - Cel mai simplu: imparte toate valorile la 2
      - Shiftare la dreapta
  - Procesul invers nu reface imaginea la starea ei anterioara
    - Pierdere de date prin compresie

1150	39	-43	-10	26	-83	11	41
-81	-3	115	-73	-6	-2	22	-5
14	-11	1	-42	26	-3	17	-38
2	-61	-13	-12	36	-23	-18	5
44	13	37	-4	10	-21	7	-8
36	-11	-9	-4	20	-28	-21	14
-19	-7	21	-6	3	3	12	-21
-5	-13	-11	-17	-4	-1	7	-4

Imagine dupa DCT

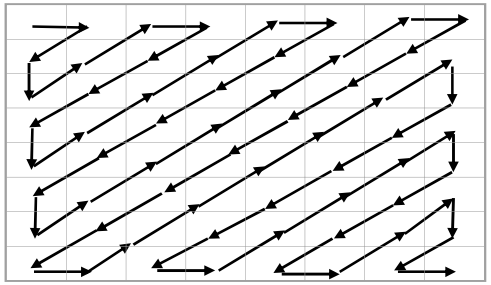
Imparte fiecare valoare din matrice cu 8

144	5	-5	-1	3	-10	1	5
-10	0	14	-9	-1	0	3	-1
2	-1	0	-5	3	0	2	-5
0	-8	-2	-2	5	-3	-2	1
6	2	5	-1	1	-3	1	-1
5	-1	-1	-1	3	-4	-3	2
-2	-1	3	-1	0	0	2	-3
-1	-2	-1	-2	-1	0	1	-1

Imagine dupa compresie

Huffman encoding

- Serializeaza blocurile de 8x8 pixeli
  - Matricea este convertita intr-o lista simplu inlantuita prin algoritmul de zig-zag de mai jos



- Codifica valorile din lista prin metoda Huffman
  - Pixelii cu frecventa cea mai mare au codul binar cel mai scurt
  - Codurile binare lungi sunt atribuite pixelilor cu frecventa mica
- Fiecare pixel din lista este inlocuit cu reprezentarea lui binara
  - Lista devine mult mai scurta -> compresie

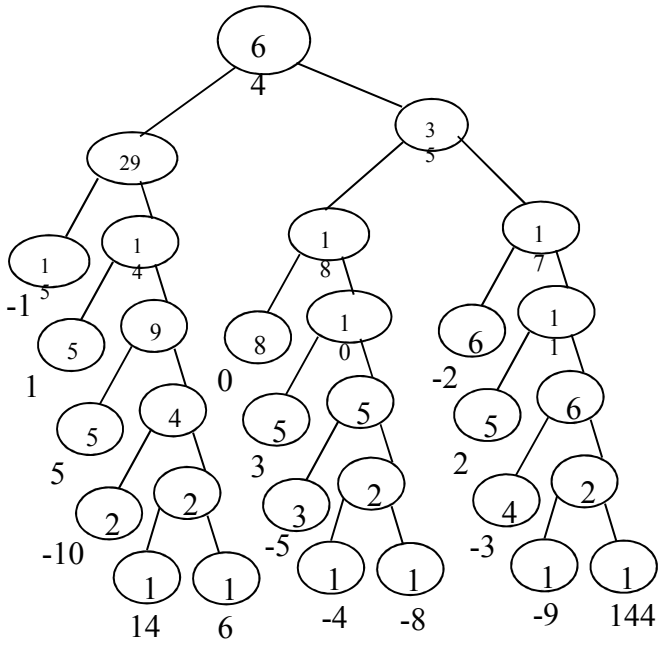
Exemplu de codare Huffman

- Frecventele pixelilor din stanga
  - Valoarea –1 apare de 15 ori
  - Valoarea 14 apare o data
- Constrieste arborele Huffaman de jos in sus
  - Creeaza cate un nod frunza pentru fiecare pixel si atribuie-i valoarea frecventei
  - Creeaza un nod intern ca parinte a doua noduri frunza
    - Valoarea lui = suma frecventelor nodurilor frunza
  - Repeta pana cand arborele binar este complet
- Parcurge arborele in adancime pentru a afla codul fiecarui pixel
  - 0 – stanga, 1- dreapta
- Codurile Huffman sunt reversibile
  - Nici un cod nu este prefixul altui cod

Frecventa pixelilor

-1	15x
0	8x
-2	6x
1	5x
2	5x
3	5x
5	5x
-3	4x
-5	3x
-10	2x
144	1x
-9	1x
-8	1x
-4	1x
6	1x
14	1x

Arbore Huffman



Coduri Huffman

-1	00
0	100
-2	110
1	010
2	1110
3	1010
5	0110
-3	11110
-5	10110
-10	01110
144	111111
-9	111110
-8	101111
-4	101110
6	011111
14	011110

Arhivarea in memorie

- Inregistreaza adresa de start si marimea imaginii
  - Se poate modela ca o lista inlantuita
- O varianta de arhivare a imaginilor:
  - Daca numarul maxim de imagini este N:
    - Aloca memorie pentru N adrese si N variabile pentru marimea imaginii
    - Calculeaza adresa de memorie pentru imaginea urmatoare
    - Initializeaza cele N adrese de imagine si variabile de marime la 0
    - Seteaza pointerul global al memoriei la N x 4
      - Presupunem ca variabilele declarate anterior nu ocupa mai mult de N x 4 octeti
    - Prima imagine poate fi salvata de la adresa N x 4
    - Pointerul global de memorie = N x 4 + (marimea imaginii comprimate)
- Cerintele de memorie depind de N, marimea imaginilor si a factorului de compresie

Descarcarea pe PC

- Atunci cand suntem conectati la PC si este primita comanda de download
  - Citeste imaginile din memorie
  - Transmite datele serial folosind interfata UART
  - In timpul transmisiei
    - Reseteaza pointerii, variabilele de marime a imaginii si pointerul global de memorie daca este cazul

Specificarea Cerintelor

- Cerintele de sistem – ce trebuie sa faca sistemul?
  - Cerinte ne-functionale
    - Constrangeri de proiectare (e.g., “trebuie sa consume 0.001W sau mai putin”)
  - Cerinte functionale
    - Comportamentul sistemului (e.g., “iesirea X trebuie sa fie Y * 2”)
  - Cerintele initiale pot sa fie foarte generale (vin de la tipii de la marketing)
    - E.g., un document scurt care exprima nevoia pentru o camera digitala care:
      - Capteaza si stocheaza 50 sau mai putine imagini si sa le transmita la PC
      - Costul sa nu depaseasca 100$
      - Are o durata de viata a bateriei cat mai mare,
      - Va avea un volum de vanzari de 200.000 buc. daca intra pe piata < 6 luni,
      - 100.000 daca intra pe piata intre 6 si 12 luni
      - Vanzari nesemnificative peste 12 luni

Specificatii ne-functionale

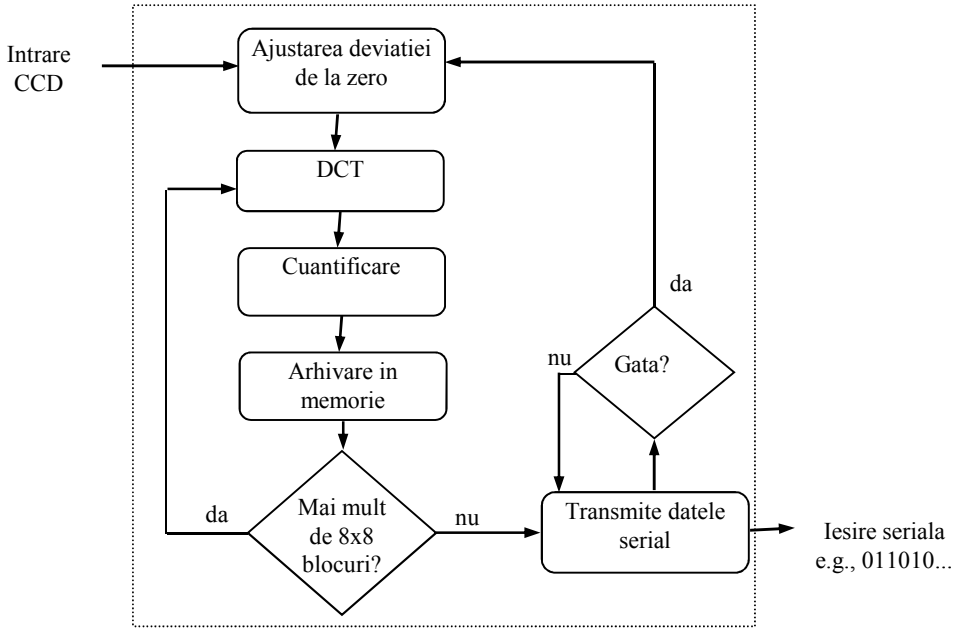
- Metrici de design bazate pe specificatiile anterioare
  - **Performanta:** timpul necesar pentru prelucrarea unei imagini
  - **Marime**: numarule de porti logice sau tranzistoare din circuitul integrat
  - **Consum**: estimarea consumului mediu de energie electrica in timpul functionarii normale
  - **Energie**: timp efectiv de viata
- Sunt metrici afectate de constrangeri
  - Valorile **trebuie** sa fie sub (sau deasupra) anumitor valori de prag
- Sunt metrici de optimizare
  - Imbunatatirea lor => imbunatatirea directa a calitatii produsului

Specificatii ne-functionale (cont.)

- Performanta
  - Trebuie sa proceseze imaginile suficient de repede ca sa fie util
  - 1 secunda este un timp acceptabil
    - Mai incet de atat ar crea neplaceri utilizatorului
    - Mai rapid – nu este necesar pentru un produs low-end
- Dimensiunile
  - Trebuie sa foloseasca circuite electronice care sa incapa intr-o carcasa de dimensiuni rezonabile
  - Metrica supusa la constrangeri si care se preteaza la optimizari
    - Ex: maxim 200.000 de porti logice pe chip, dar produsul este mai ieftin daca numarul lor este mai mic
- Cerinte de putere
  - Nu poate sa opereze peste o anumita temperatura (nu pot sa integrez elemente de racire)
  - Metrica cu constrangeri
- Consum de energie
  - Reducerea puterii sau timpului de procesare inseamna reducerea consumului
  - Se preteaza la optimizari: vreau ca timpul de viata al bateriei sa fie cat mai mare

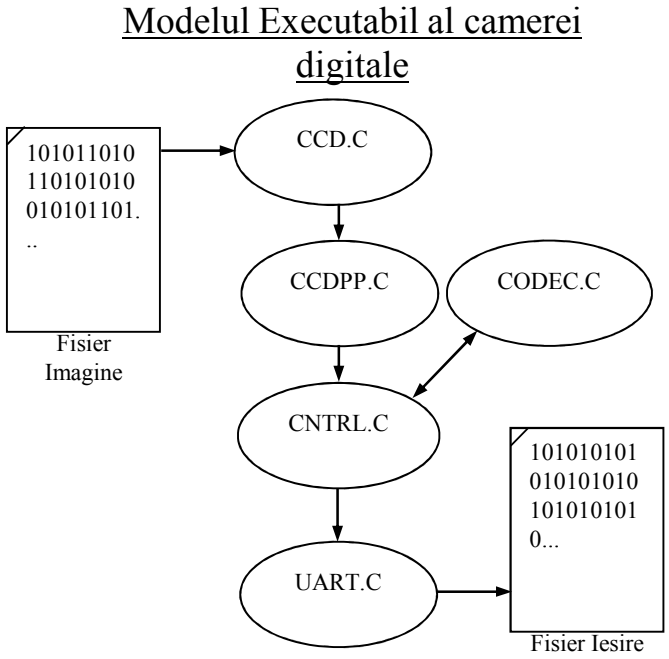
Specificatii functionale

- Utilizam diagrame de stari pentru a sparge specificatiile anterioare in mai multe unitati functionale independente.
- Fiecare functie poate fi descrisa in limbaj natural
- Alegerea tipului de procesor sau a altor circuite specifice se va face la un pas ulterior



Specificatii functionale extinse

- Pasul urmator: rafinarea specificatiilor anterioare intr-un model executabil
- Fiecare functie poate fi descrisa prin cod
- Poate furniza ceva indicii despre functionalitatea sistemului
  - Se poate face profiling pentru a determina task-urile computing-intensive
- Obtine o iesire de test care este folosita pentru a testa functionalitatea si performantele intregului sistem



Modulul CCD

- Simuleaza un CCD real
- *CcdInitialize* preia numele fisierului de intrare
- *CcdCapture* citeste “imaginea” din fisier
- *CcdPopPixel* serializeaza pixelii din imagine

```
#include <stdio.h>
#define SZ_ROW       64
#define SZ_COL      (64 + 2)
static FILE *imageFileHandle;
static char buffer[SZ_ROW][SZ_COL];
static unsigned rowIndex, colIndex;
```

```
char CcdPopPixel(void) {
    char pixel;
    pixel = buffer[rowIndex][colIndex];
    if( ++colIndex == SZ_COL ) {
        colIndex = 0;
        if( ++rowIndex == SZ_ROW ) {
            colIndex = -1;
            rowIndex = -1;
        }
    }
    return pixel;
}
```

```
void CcdInitialize(const char *imageFileName) {
    imageFileHandle = fopen(imageFileName, "r");
    rowIndex = -1;
    colIndex = -1;
}
```

```
void CcdCapture(void) {
    int pixel;
    rewind(imageFileHandle);
    for(rowIndex=0; rowIndex<SZ_ROW; rowIndex++) {
        for(colIndex=0; colIndex<SZ_COL; colIndex++) {
            if( fscanf(imageFileHandle, "%i", &pixel) == 1 ) {
                buffer[rowIndex][colIndex] = (char)pixel;
            }
        }
    }
    rowIndex = 0;
    colIndex = 0;
}
```

Modulul CCDPP (CCD PreProcessing)

- Efectueaza corectia de eroare
- *CcdppCapture* foloseste *CcdCapture* si *CcdPopPixel* pentru a obtine imaginea
- Corectia erorii se efectueaza dupa citirea fiecarui rand

```
void CcdppCapture(void) {
    char bias;
    CcdCapture();
    for(rowIndex=0; rowIndex<SZ_ROW; rowIndex++) {
        for(colIndex=0; colIndex<SZ_COL; colIndex++) {
            buffer[rowIndex][colIndex] = CcdPopPixel();
        }
        bias = (CcdPopPixel() + CcdPopPixel()) / 2;
        for(colIndex=0; colIndex<SZ_COL; colIndex++) {
            buffer[rowIndex][colIndex] -= bias;
        }
    }
    rowIndex = 0;
    colIndex = 0;
}
```

```
#define SZ_ROW      64
#define SZ_COL      64
static char buffer[SZ_ROW][SZ_COL];
static unsigned rowIndex, colIndex;
```

```
void CcdppInitialize() {
    rowIndex = -1;
    colIndex = -1;
}
```

```
char CcdppPopPixel(void) {
    char pixel;
    pixel = buffer[rowIndex][colIndex];
    if( ++colIndex == SZ_COL ) {
        colIndex = 0;
        if( ++rowIndex == SZ_ROW ) {
            colIndex = -1;
            rowIndex = -1;
        }
    }
    return pixel;
}
```

Modulul UART

- De fapt, jumatate de UART
  - Doar transmisie, fara receptie
- *UartInitialize* primeste numele fisierului care trebuie transmis
- *UartSend* transmite (scrie in fisierul de iesire) octetii serializati ai imaginii

```
#include <stdio.h>
static FILE *outputFileHandle;
void UartInitialize(const char *outputFileName) {
    outputFileHandle = fopen(outputFileName, "w");
}
void UartSend(char d) {
    fprintf(outputFileHandle, "%i\n", (int)d);
}
```

Modulul CODEC

- Modeleaza codarea FDCT
- *ibuffer* tine blocul original 8 x 8
- *obuffer* contine blocul codat 8 x 8
- *CodecPushPixel* apelata de 64 de ori pentru a umple *ibuffer* cu blocul neprocesat
- *CodecDoFdct* apelata o data pentru a procesa blocul 8 x 8
  - Explicata in slide-ul urmator
- *CodecPopPixel* apelata de 64 de ori pentru a serializa blocul codat din *obuffer*

```
static short ibuffer[8][8], obuffer[8][8], idx;

void CodecInitialize(void) { idx = 0; }
```

```
void CodecPushPixel(short p) {
    if( idx == 64 ) idx = 0;
    ibuffer[idx / 8][idx % 8] = p; idx++;
}
```

```
void CodecDoFdct(void) {
    int x, y;
    for(x=0; x<8; x++) {
        for(y=0; y<8; y++)
            obuffer[x][y] = FDCT(x, y, ibuffer);
    }
    idx = 0;
}
```

```
short CodecPopPixel(void) {
    short p;
    if( idx == 64 ) idx = 0;
    p = obuffer[idx / 8][idx % 8]; idx++;
    return p;
}
```

CODEC (cont.)

- Implementarea formulei de calcul a FDCT

$$G_{u,v} = \alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} g_{x,y}\cos\left[\frac{\pi}{8}\left(x+\frac{1}{2}\right)u\right]\cos\left[\frac{\pi}{8}\left(y+\frac{1}{2}\right)v\right]$$

- Sunt doar 64 de intrari posibile pentru COS, asa ca putem folosi un tabel pentru a scapa de niste timpi de procesare
- *FDCT* desface bucla interioara a sumei si implementeaza suma exterioara prin doua bucle for consecutive

```
static const short COS_TABLE[8][8] = {
    { 32768,  32138,  30273,  27245,  23170,  18204,  12539,   6392 },
    { 32768,  27245,  12539,  -6392, -23170, -32138, -30273, -18204 },
    { 32768,  18204, -12539, -32138, -23170,   6392,  30273,  27245 },
    { 32768,   6392, -30273, -18204,  23170,  27245, -12539, -32138 },
    { 32768,  -6392, -30273,  18204,  23170, -27245, -12539,  32138 },
    { 32768, -18204, -12539,  32138, -23170,  -6392,  30273, -27245 },
    { 32768, -27245,  12539,   6392, -23170,  32138, -30273,  18204 },
    { 32768, -32138,  30273, -27245,  23170, -18204,  12539,  -6392 }
};
```

```
static short ONE_OVER_SQRT_TWO = 23170;
static double COS(int xy, int uv) {
     return COS_TABLE[xy][uv] / 32768.0;
}
static double C(int h) {
     return h ? 1.0 : ONE_OVER_SQRT_TWO / 32768.0;
}
```

```
static int FDCT(int u, int v, short img[8][8]) {
    double s[8], r = 0; int x;
    for(x=0; x<8; x++) {
     s[x] = img[x][0] * COS(0, v) + img[x][1] * COS(1, v) +
            img[x][2] * COS(2, v) + img[x][3] * COS(3, v) +
            img[x][4] * COS(4, v) + img[x][5] * COS(5, v) +
            img[x][6] * COS(6, v) + img[x][7] * COS(7, v);
    }
    for(x=0; x<8; x++) r += s[x] * COS(x, u);
    return (short)(r * .25 * C(u) * C(v));
}
```

Modulul CNTRL (controller)

- Este inima sistemului
- *CntrlInitialize*
- *CntrlCaptureImage* foloseste modulul CCDPP pentru a plasa imaginea intr-un buffer
- *CntrlCompressImage* sparge buffer-ul de 64x64 pixeli in blocuri 8x8 si aplica FDCT pe fiecare bloc in parte folosid modulul CODEC
  - Mai si cuantifica datele
- *CntrlSendImage* transmite imaginea codata serial prin intermediul modulului UART

```
void CntrlCaptureImage(void) {
    CcdppCapture();
    for(i=0; i<SZ_ROW; i++)
        for(j=0; j<SZ_COL; j++)
            buffer[i][j] = CcdppPopPixel();
}
```

```
#define SZ_ROW          64
#define SZ_COL          64
#define NUM_ROW_BLOCKS  (SZ_ROW / 8)
#define NUM_COL_BLOCKS  (SZ_COL / 8)
static short buffer[SZ_ROW][SZ_COL], i, j, k, l, temp;
void CntrlInitialize(void) {}
```

```
void CntrlSendImage(void) {
    for(i=0; i<SZ_ROW; i++)
        for(j=0; j<SZ_COL; j++) {
            temp = buffer[i][j];
            UartSend(((char*)&temp)[0]);    /* send upper byte */
            UartSend(((char*)&temp)[1]);    /* send lower byte */
        }
    }
}
```

```
void CntrlCompressImage(void) {
    for(i=0; i<NUM_ROW_BLOCKS; i++)
        for(j=0; j<NUM_COL_BLOCKS; j++) {
            for(k=0; k<8; k++)
                for(l=0; l<8; l++)
                    CodecPushPixel(
                        (char)buffer[i * 8 + k][j * 8 + l]);
            CodecDoFdct();/* part 1 - FDCT */
            for(k=0; k<8; k++)
                for(l=0; l<8; l++) {
                    buffer[i * 8 + k][j * 8 + l] = CodecPopPixel();
                    /* part 2 - quantization */
                    buffer[i*8+k][j*8+l] >>= 6;
                }
        }
}
```

Asamblarea toturor modulelor

- *Main* initializeaza toate modulele, apoi foloseste modulul CNTRL pentru a capta, comprima si a transmite o imagine
- Acest model de sistem poate fi folosit pentru o gama larga de experimente inaintea implementarii propriu-zise a hardware-ului
  - Depanarea se face mult mai usor la acest nivel decat mai tarziu.

```
int main(int argc, char *argv[]) {
    char *uartOutputFileName = argc > 1 ? argv[1] : "uart_out.txt";
    char *imageFileName = argc > 2 ? argv[2] : "image.txt";
    /* initialize the modules */
    UartInitialize(uartOutputFileName);
    CcdInitialize(imageFileName);
    CcdppInitialize();
    CodecInitialize();
    CntrlInitialize();
    /* simulate functionality */
    CntrlCaptureImage();
    CntrlCompressImage();
    CntrlSendImage();
}
```

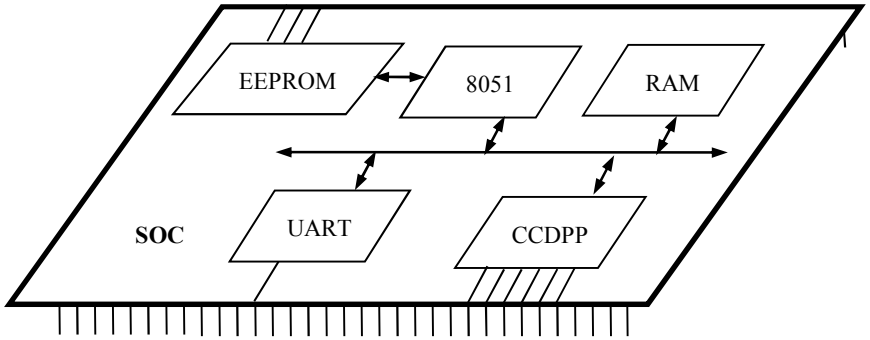
Proiectarea sistemului

- Determina arhitectura sistemului
  - Procesoare
    - Orice combinatie de procesoare single-purpose sau general-purpose
  - Memorii, magistrale de comunicatie
- Mapeaza functionalitatea sistemului pe arhitectura selectata
  - Un procesor poate avea mai multe functii
  - O functie poate fi distribuita pe mai multe procesoare
- Implementare
  - Alegerea unei anumite arhitecturi
  - Spatiul solutiilor contine setul tuturor implementarilor posibile
- De unde incepem?
  - Procesor low-end, general purpose conectat la o memorie flash
    - Toata functionalitatea este mapata in softul care ruleaza pe acel procesor
    - De obicei satisface toate constrangerile de consum, dimensiuni si time-to market
    - Daca una din constrangeri nu este satisfacuta, atunci implementarile viitoare pot sa:
      - Foloseasca un alt procesor single-purpose pentru procesele time-critical
      - Rescrie specificatiile functionale

Implementarea 1: Un singur Microcontroller

- Un procesor low-end adecvat ar putea fi Intel 8051
- Costul total al chipului ~ $5
- Consum de energie sub 200mW
- Time-to-market de aproximativ 3 luni
- DAR, o imagine pe secunda este peste posibilitatile lui
  - 12 MHz, 12 cicli instructiune
    - Executa un milion de instructiuni pe secunda
  - *CcdppCapture* are bucle imbricate de 4096 (64 x 64) iteratii
  - ~100 de instructiuni de asamblare pt fiecare iteratie
    - 409,000 (4096 x 100) instructiuni pe imagine
    - Aproape jumatate de milion de instructiuni doar ca sa citeasca o imagine
  - Daca mai adaugam si DCT si codificarea Huffman, sarim cu mult peste tinta de un milion de instructiuni

Implementarea 2: Microcontroller si CCDPP

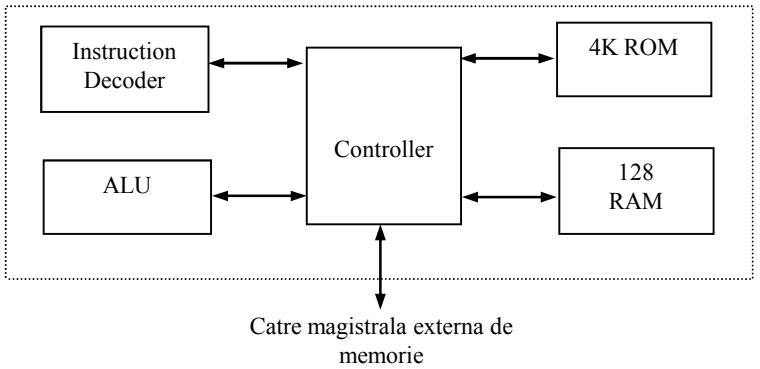


- Functia CCDPP este implementata pe un procesor dedicat
- Imbunatateste performantele – mai putini cicli de procesor
  - Mareste costul si time-to-market
  - Usor de implementat
    - Magistrala simpla
    - Controller de bus cu putine stari
- UART simplu: usor de implementat intr-un alt procesor dedicat
- Se adauga memorie EEPROM si RAM pentru stocarea datelor

Microcontroller

- Versiunea sintetizabila a lui Intel 8051 este disponibila
  - Scrisa in VHDL
  - Implementata la nivel de register transfer (RTL)
- Face fetch-ul unei instructiuni din ROM
- Decodifica cu Instruction Decoder
- ALU executa operatiile aritmetice
  - Registrele sursa si destinatie sunt mapate in RAM
- Instructiuni speciale pentru load si store extern
- Exista programe speciale care genereaza cod VHDL din cod simplu C/C++

Diagrama bloc a nucleului Intel 8051



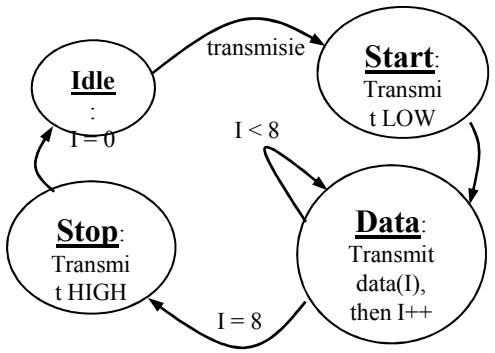
UART

- UART ramane in modul idle daca nu este folosit
  - UART este folosit atunci cand 8051 executa operatia de store cu adresa registrului de date al acestuia

FSM:

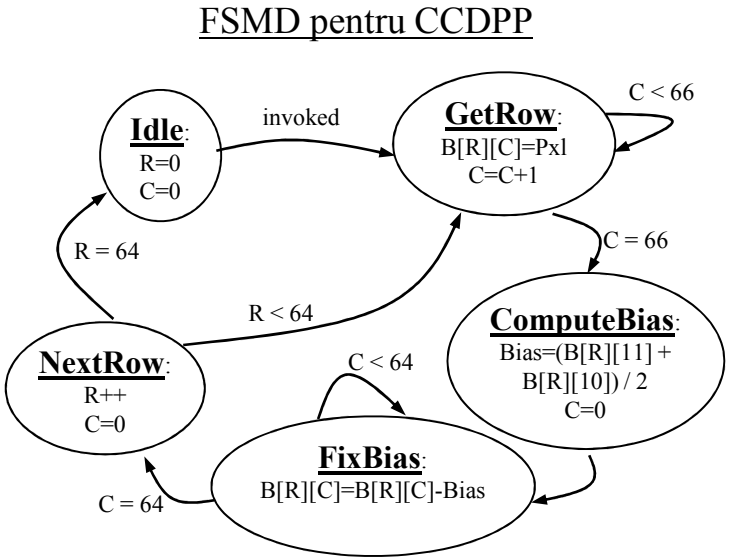
- Starea de start transmite 0 indicand startul transmisiei apoi trece in starea Data
- Starea Data trimite 8 biti serial apoi trece in starea Stop
- Stop transmite 1, indicand terminarea transmisiei apoi trece inapoi in modul Idle

FSMD pentru UART



CCDPP

- Implementarea hardware a corectiei de eroare de zero
- Interactioneaza cu chipul CCD extern
  - Chipul CCD nu poate fi inclus in SoC-ul nostru din ratiuni de proiectare si interferente
- Buffer intern B, mapat de 8051
- R, C sunt indecsii liniilor, respectiv coloanelor lui B
- Starea GetRow aduce un rand din CCD in *B*
  - 66 octeti: 64 pixeli + 2 pixeli obturati
- Starea ComputeBias calculeaza eroarea de zero pentru randul curent si o memoreaza in *Bias*
- Starea FixBias itereaza pe valorile din rand si scade *Bias* din fiecare
- NextRow trece la GetRow sau la Idle daca nu mai sunt randuri de citit



Software

- Modelul de sistem descris anterior poate furniza majoritatea codului
  - Ierarhia modulelor, numele de proceduri si programul principal raman neschimbate
- Codul pentru UART si CCDPP trebuie rescris
  - Se doreste o reutilizare cat mai mare a codului

Cod original din modelul de sistem

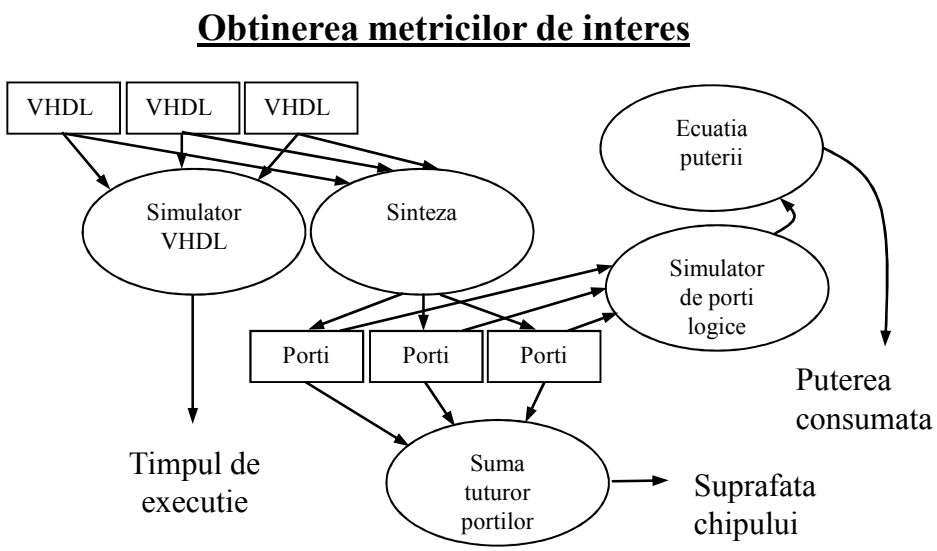
```
#include <stdio.h>
static FILE *outputFileHandle;
void UartInitialize(const char *outputFileName) {
    outputFileHandle = fopen(outputFileName, "w");
}
void UartSend(char d) {
    fprintf(outputFileHandle, "%i\n", (int)d);
}
```

Cod UART rescris

```
static unsigned char xdata U_TX_REG _at_ 65535;
static unsigned char xdata U_STAT_REG _at_ 65534;
void UARTInitialize(void) {}
void UARTSend(unsigned char d) {
    while( U_STAT_REG == 1 ) {
        /* busy wait */
    }
    U_TX_REG = d;
}
```

Analiza sistemului

- Intregul SoC poate fi testat intr-un simulator de VHDL
  - Se poate face testarea functionalitatii
  - Masuratori de performanta (cicli de ceas pe imagine)
- Descrierea sistemului la nivel de porti logice obtinuta in urma sintezei
  - Simuleaza modelul la nivel de porti logice pentru a obtine date despre performanta sistemului
  - Numarul de porti logice afecteaza performantele sistemului



Implementarea 2: Microcontroller si CCDPP

- Analiza implementarii 2
  - Timpul total de executie pentru procesarea unei imagini:
    - 9.1 secunde
  - Puterea consumata:
    - 0.033 watt
  - Consumul de energie:
    - 0.30 jouli (9.1 s x 0.033 watt)
  - Aria totala a chipului:
    - 98,000 porti logice

Implementarea 3: Microcontroller si CCDPP/Fixed-Point DCT

- 9.1 secunde tot nu satisface constrangerile de performanta stabilite anterior (1 secunda)
- Cel mai usor de imbunatatit este modulul DCT
  - Din analiza implementarii 2, se poate vedea ca procesorul aloca un numar foarte mare de cicli pentru efectuarea DCT
  - Se poate proiecta o unitate separata, ca pentru CCDPP
    - Complexitatea este mai mare, efortul de proiectare este substantial
  - O alta abordare ar fi sa optimizam timpii de procesare prin modificarea comportamentului

Costurile DCT in virgula mobila

- Costurile asociate operatiei in v.m.
  - DCT necesita ~260 operatii in v.m. pt fiecare pixel
  - 4096 (64 x 64) pixeli pe imagine
  - 1 milion de op in v.m. pentru fiecare imagine
  - Intel 8051 nu are suport hardware pentru v.m.
    - Compilatorul trebuie sa emuleze
      - Pentru fiecare operatie este generata o procedura
      - Fiecare procedura foloseste zeci de instructiuni cu intregi
  - Rezulta > 10 milioane de operatii cu intregi pe imagine
  - Procedurile “ingrasa”si codul compilat
- Aritmetica in virgula fixa poate sa ne scape de probleme

Aritmetica in virgula fixa

- Numerele reale sunt reprezentate ca intregi
  - Un numar constant de biti din intreg reprezinta partea fractionala a numarului real
    - Cu cat sunt mai multi biti, cu atat e mai precisa reprezentarea
  - Ceilalti biti reprezinta partea intreaga a numarului real
- Trecerea din real in virgula fixa:
  - Inmulteste valoarea reala cu 2 ^ (# nr de biti stabiliti pt partea fractionala)
  - Rotunjeste la cel mai apropiat intreg
  - E.g., 3.14 reprezentat ca un intreg de 8-biti cu 4 biti pt fractie
    - 2^4 = 16
    - 3.14 x 16 = 50.24 ≈ 50 = 00110010
    - 16 (2^4) valori posibile pt fractie, fiecare reprezinta 0.0625 (1/16)
    - Ultimii 4 biti (0010) = 2
    - 2 x 0.0625 = 0.125
    - 3(0011) + 0.125 = 3.125 ≈ 3.14 (mai multi biti pt fractie maresc precizia)

Operatii in virgula fixa

- Adunarea
  - Aduna reprezentarile binare
  - E.g., 3.14 + 2.71 = 5.85
    - 3.14 → 50 = 00110010
    - 2.71 → 43 = 00101011
    - 50 + 43 = 93 = 01011101
    - 5(0101) + 13(1101) x 0.0625 = 5.8125 ≈ 5.85
- Inmultirea
  - Inmulteste reprezentarile binare
  - Shifreaza rezultatul cu numarul de biti din partea fractionara
  - E.g., 3.14 * 2.71 = 8.5094
    - 50 * 43 = 2150 = 100001100110
    - >> 4 = 10000110
    - 8(1000) + 6(0110) x 0.0625 = 8.375 ≈ 8.5094

Implementarea in v.f. a modulului CODEC

- COS_TABLE contine reprezentarea in v.f. pe 8 biti a valorilor cosinusului
- 6 biti alocati pentru partea fractionala
- Rezultatul inmultirilor este shiftat dreapta cu 6 pozitii

```
static const char code COS_TABLE[8][8] = {
    {   64,   62,   59,   53,   45,   35,   24,   12 },
    {   64,   53,   24,  -12,  -45,  -62,  -59,  -35 },
    {   64,   35,  -24,  -62,  -45,   12,   59,   53 },
    {   64,   12,  -59,  -35,   45,   53,  -24,  -62 },
    {   64,  -12,  -59,   35,   45,  -53,  -24,   62 },
    {   64,  -35,  -24,   62,  -45,  -12,   59,  -53 },
    {   64,  -53,   24,   12,  -45,   62,  -59,   35 },
    {   64,  -62,   59,  -53,   45,  -35,   24,  -12 }
};
```

```
static unsigned char C(int h) { return h ? 64 : ONE_OVER_SQRT_TWO;}
static int F(int u, int v, short img[8][8]) {
    long s[8], r = 0;
    unsigned char x, j;
    for(x=0; x<8; x++) {
        s[x] = 0;
        for(j=0; j<8; j++)
            s[x] += (img[x][j] * COS_TABLE[j][v] ) >> 6;
    }
    for(x=0; x<8; x++) r += (s[x] * COS_TABLE[x][u]) >> 6;
    return (short)((((r * (((16*C(u)) >> 6) *C(v)) >> 6)) >> 6) >> 6);
}
```

```
static const char ONE_OVER_SQRT_TWO = 5;
static short xdata inBuffer[8][8], outBuffer[8][8], idx;

void CodecInitialize(void) { idx = 0; }
```

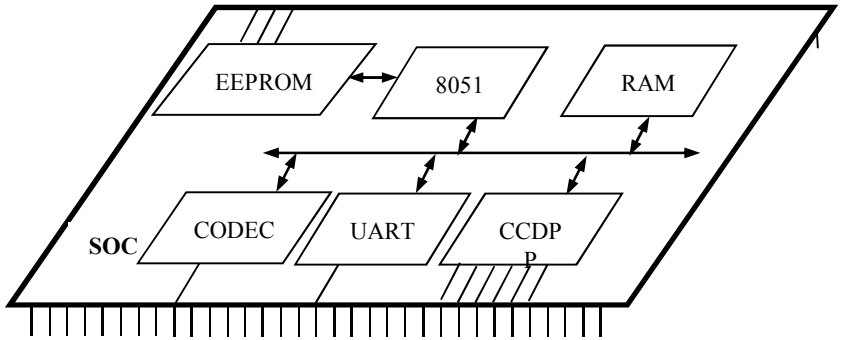
```
void CodecPushPixel(short p) {
    if( idx == 64 ) idx = 0;
    inBuffer[idx / 8][idx % 8] = p << 6; idx++;
}
```

```
 void CodecDoFdct(void) {
     unsigned short x, y;
     for(x=0; x<8; x++)
         for(y=0; y<8; y++)
             outBuffer[x][y] = F(x, y, inBuffer);
     idx = 0;
 }
```

Implementarea 3: Microcontroller si CCDPP/Fixed-Point DCT

- Analiza implementarii 3:
  - Foloseste aceleasi tehnici de analiza ca si pentru implementarea 2
  - Timpul total de procesare a unei imagini:
    - 1.5 secunde
  - Puterea consumata:
    - 0.033 watt (identic cu 2)
  - Consumul de energie:
    - 0.050 jouli (1.5 s x 0.033 watt)
    - Durata de biata a bateriei e de 6 ori mai mare!!
  - Suprafata totala a chipului:
    - 90,000 porti logice
    - Cu 8,000 de porti mai putin (mai putina memorie)

Implementarea 4: Microcontroller si CCDPP/DCT



- Performantele variantei anterioarea sunt bune, dar nu suficient de bune.
- Trebuie sa implementez CODEC-ul in hardware
  - Procesor dedicat executiei DCT pe blocuri 8 x 8 pixeli

Implementarea 4: Microcontroller si CCDPP/DCT

- Analiza implementarii 4:
  - Timpul total pentru procesarea unei imagini:
    - 0.099 secunde (cu mult sub o secunda)
  - Puterea consumata:
    - 0.040 watt
    - Mai mare decat variantele 2 si 3 pentru ca SoC are inca un procesor
  - Energia consumata:
    - 0.00040 jouli (0.099 s x 0.040 watt)
    - Durata de viata a bateriei de 12 ori mai mare decat prima varianta!!
  - Suprafata totala a chipului:
    - 128,000 porti logice
    - Crestere substantiala fata de celelalte variante

Concluzii

	Implementarea 2	Implementarea 3	Implementarea 4
Performanta (s)	9,1	1,5	0,099
Putere (W)	0,033	0,033	0.040
Arie(nr porti)	98,000	90,000	128,000
Energie (joule)	0.30	0.050	0.0040

- Implementarea 3
  - Destul de apropiata de cerinte
  - Mai ieftina
  - Mai putin timp pentru proiectare/realizare
- Implementarea 4
  - Performante marite de timp si consum de energie
  - Mai scumpa decat 3 si timp de dezvoltare mai lung (poate rata time-to-market)

- Care implementare este mai buna?

Concluzii

- Exemplu de camera digitala
  - Specificatii scrise in limbaj natural si in limbaj de programare
  - Metrici folosite in design: performanta, consum, dimensiuni
- Mai multe implementari
  - Microcontroller: prea incet
  - Microcontroller si coprocessor: mai bun, dar tot prea lent
  - Aritmetica in v.f.: aproape suficient de rapid
  - Coprocesor aditional pentru compresie: foarte rapid dar costisitor si greu de proiectat
  - Compromis intre hw/sw