Long-distance Lossy Livestream - Ulmeanu Vlad-Adrian 335CA

Introducere

Proiectul are ca scop crearea unui modul extern capabil sa transmita un flux video de la distante relativ mari, in dezavantajul calitatii imaginii si a numarului scazut de cadre pe secunda.

Laboratoare folosite: 1 (USART), 2 (Intreruperi), 3 (Timere) si 5 (SPI).

Descriere generală

- Periodic, modulul extern va face o poza, o va compresa si o va transmite modulului local.
- Algoritmul de compresie se uita la valorile pixelilor din imagine. Daca sunt indeajuns de apropriate, le aproximeaza pe toate cu o valoare medie. Altfel, sparge imaginea in patru cadrane si repeta algoritmul.

Schema bloc:

Hardware Design

Modulul extern este format dintr-un:

- ESP32-WROOM-32U,
- un transceiver LoRa SX1278,
- un modul camera ESP32-CAM
- si o baterie.

Modulul local este format din:

- alt ESP32-WROOM-32U,
- alt transceiver LoRa SX1278
- si un laptop pentru display.

Schema:

pmproj_uva_schematic.pdf

Cele doua transceivere LoRa functioneaza in mod half-duplex. In acest design, transceiver-ul de pe modulul local trebuie sa verifice CRC-ul pachetului si sa transmita inapoi un tip de ACK.

Pinul `DIO0` de pe LoRa genereaza intreruperi. Cele trei care ne intereseaza sunt:

- `onReceive`. Un pachet nou este disponibil in bufferul LoRa.
- `onTxDone`. Pachetul din buffer a fost trimis. Bufferul poate fi refolosit. Nu se garanteaza ca celalalt LoRa a primit pachetul.
- `onCrcError`. A venit un pachet nou, insa CRC-ul nu este corect. Continutul pachetului nu este disponibil.

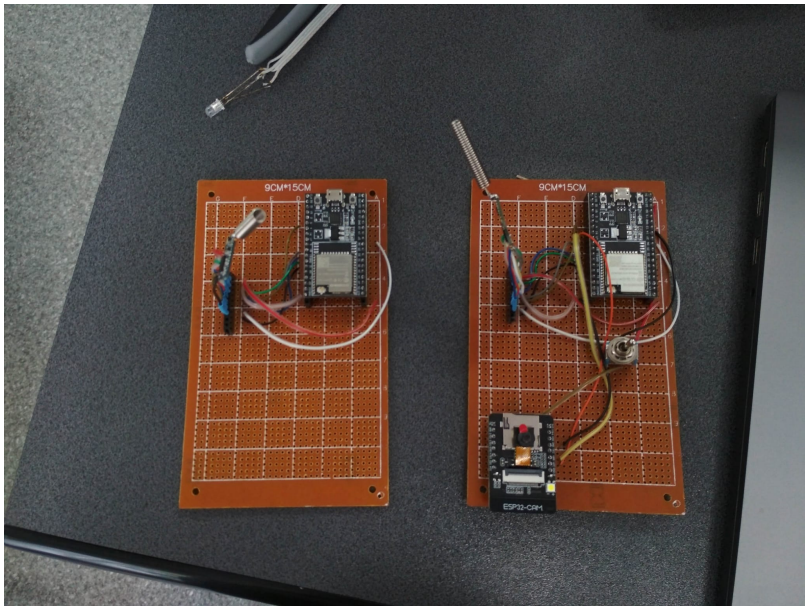
Exista un bug documentat, dar inca prezent in cea mai recenta versiune a bibliotecii: nu apare intreruperea lui `onTxDone`, chiar daca bufferul a fost golit:

```
int LoRaClass::endPacket(bool async) {
...
    if (_onTxDone && (readRegister(REG_IRQ_FLAGS) & IRQ_TX_DONE_MASK) ==
IRQ_TX_DONE_MASK) {
        handleDio0Rise();
...
}
```

`onCrcError` nu este implementat deloc. Tratamentul standard implica aruncarea silentioasa a pachetelor cu CRC-ul gresit. Am pus interrupt-ul in biblioteca (copiat comportamentul de la `onTxDone` si adaugat un `else` ca sa previn drop-ul):

```
void LoRaClass::handleDio0Rise() {
    ...
    if ((irqFlags & IRQ_PAYLOAD_CRC_ERROR_MASK) == 0) {
        ...
    } else if (_onCrcError) {
        _onCrcError();
    }
}
```

Receiver-ul si sender-ul trecute pe placi de prototipare:



Software Design

Design sumar al algortimului de compresie:

- Pozele necompresate au rezolutia de `256 x 256`.
- Presupunem ca pixelii unei poze ajung in memorie sub forma unei matrice `v[0..255][0.255]`.
- Pentru simplificare, presupunem ca folosim un singur canal de culoare.
- Vrem sa calculam `d[i][j][l]` si `D[i][j][L]` pentru anumiti `i, j` din `[0..255]` si `l` putere a lui 2.
- `d[i][j][l]` reprezinta valoarea minima a unui pixel din careul `[i..i+l-1] x [j..j+l-1]`.
- `D[i][j][l]` reprezinta valoarea maxima a unui pixel din careul `[i..i+l-1] x [j..j+l-1]`.
- Fie `scor` o functie care accepta parametrii `l, minVal, maxVal`. Daca `scor(l, d[i][j][l], D[i][j][l]) ⇐ thresh` (o constanta), nu mai sparg careul in patru cadrane si aproximez toti pixelii din careu cu media valorilor lor.
- Altfel, sparg careul `(i, j, l)` in patru cadrane:
  - `(i, j, l/2)`,
  - `(i, j + l/2, l/2)`,
  - `(i + l/2, j, l/2)`,
  - `(i + l/2, j + l/2, l/2)`.
- si continui recursiv algoritmul.

Matricele `d[][][]` si `D[][][]` se construiesc de la `l` mic la mare (bottom-up), apoi urmeaza recursia top-down.

Pe `ESP32-CAM` `d` si `D` sunt reprezentate liniar. Interactiunea fiu-tata este foarte similara cu cea de la reprezentarea unui heap in memorie (numarand de la `0`, daca tatal este la pozitia `i`, fiii sunt la pozitiile `2i+1` si `2i+2`).

Diferenta este ca aici un nod are 4 fii, iar fiecare nod ocupa 3 pozitii consecutive, una pentru fiecare culoare.

```
int getChild(int r, int id) {
```

```
    return ((r / 3) * 4 + id) * 3;
}

int getFather(int r) {
    return ((r / 3) - 1) / 4 * 3;
}
```

Functia scor are urmatoarea implementare:

```
dispThresh = 80

def corectorDisp(arie: int) -> float: #daca aria este prea mica, mai las din
dispThresh.
    if arie > 1000:
        return 1.0
    ex = math.exp(arie * 0.00390625) #coef pt 1024x768. cu cat e coef mai
mic, cu atat compreseaza mai tare.
    return ex / (1 + ex) #normalizare pe [0, 1] cu sigmoid.

def scor(l, minVal, maxVal): #minVal si maxVal in [0, 255].
    return (maxVal - minVal) * corectorDisp(l * l)

#spargere daca scor(l, d[i][j][l], D[i][j][l]) > dispThresh.
```

Pe ESP32-CAM coeficientii de corectie sunt tinuti direct intr-o tabela de lookup. Normalizarea pe `[0, 1]` in loc de `[0.5, 1]` produce mult mai putini pixeli cu arii diferite.

```
static float coef_corector_lookup[9] = {0.501f, 0.5039f, 0.5156f, 0.5622f,
0.7311f, 0.982f, 1.0f, 1.0f, 1.0f}; //0.00390625f
```

Simularea locala a algortimului de compresie.

`0.12%` din pixeli pastrati dupa compresie.

`0.3%` din pixeli pastrati dupa compresie.

Schema protocolului pentru trimiterea unui cadru. Exista exemple pentru CRC fail si pentru pachet pierdut.

(Concluzie ulterioara) schema protocolului a ajutat la rezolvarea bugului legat de primirea de pachete eronate daca distanta fata de laptop era putin mai mare (peste cativa metri): ACK-ul trebuia trimis imediat inapoi de catre Receiver. Lasam seriala catre laptop sa transmita octetii primiti inainte de a face asta. Cumva cuanta de timp alocata pentru asta era prea mare.

Rezultate Obţinute

Git repo.

- Cod ESP32-CAM: Link.
- Cod Sender: Link.
- Cod Receiver: Link.
- Cod Laptop: Link.

Mai jos sunt cateva poze obtinute in urma compresiei pe microcontroller:

Un zambet compresat. Paleta deplasata de culori poate fi cauzata de formatul RGB565. 808 pixeli compresati (1.2%), 2.95KB.

Unicul IDE si un portal catre cealalta lume (afara). 1174 pixeli compresati (1.8%), 4.45KB.

Biroul. Unele detalii sunt mult mai clare (marginile dintre doua culori indeajuns de diferite, fire de pe

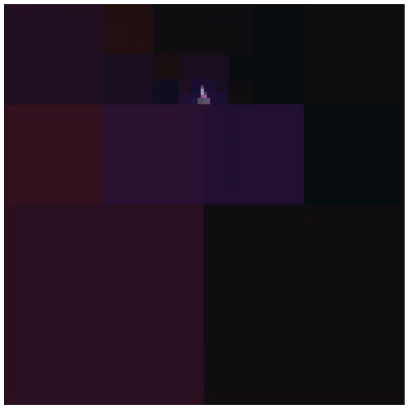
breadboard). 2107 pixeli compresati (3.2%), 8.25KB.

Daisychain ESP32-CAM —(serial)—> ESP32-WROOM —(serial)—> laptop, 115200 bps. Link video (speedup 2x)

Poza cu transmiterea prin LoRa functionala:

Test cu LoRa (speedup 4x, ~4500 bps):

Acum ca stream-ul merge la o distanta mai mare de laptop, pot sa arat poze precum cea de mai jos (luna, 2+ pereti distanta):



Concluzii

- Nu ma asteptam ca numarul de pixeli dupa compresie sa nu fie proportional cu rezolutia imaginii initiale. Se pare ca depinde mai mult de cat de detaliata este imaginea.
- Transmiterea informatiilor este clar bottleneck-ul proiectului. Compresarea imaginii este mult mai rapida.
- Transmiterea prin LoRa este cea mai grea parte din proiect. Trebuie sa investighez si setarile de putere. ~~Deocamdata un perete este de ajuns pentru a pune probleme serioase transmiterii, chiar si cu masurile de corectie implementate.~~ (din fericire am reparat bugul, explicatia este jos la Software Design)

Download

Jurnal

- 13 aprilie: achizitionare piese.
- 3 mai: descriere proiect.
- 7 mai: sumar Software Design.
- 21 mai: sumar Rezultate Obtinute.
- 23 mai: video cu modul Daisychain.
- 28 mai: finalizare cod interactiune LoRa.
- 29 mai: trecere pe PCB-uri, bugfix stream la distanta non-mica.

Bibliografie/Resurse

Export to PDF