Caro Marcos,

Achei que a última versão do VFO/BFO que usei no experimento do rádio baseado no CD2003GP é a melhor que fiz até agora, no que tange à apresentação e o comportamento da interface. Claro, pretendo evoluir, mas por hora está razoável. Ela funciona no NANO, Pro Mini etc. Retirei o erro do truncamento da Unidade para SW. Mas há um sério limite de espaço nesse display que estou usando. Tentarei usar outro display. Os que tenho aqui não são I2C. Verei como faço.

Como falei no e-mail anterior, a alteração que fiz foi para oferecer um comportamento diferente para bandas diferentes. Ou seja, para sintonizar estações de FM, por exemplo, só é necessário 1 dígito para casa decimal, passos entre 50KHz e 500KHz são mais apropriados que passos muito pequenos como 1KHz usado em SW. Já em AM, passos entre 500Hz e 50KHz, bem como 2 dígitos após a vírgula, creio que funcionam bem . Em SW, é desejável uma precisão melhor, então o passo entre 100Hz e 50KHz, bem como duas casas após a vírgula pode ser mais interessante. Então vamos a tabela para implementar comportamentos distintos por banda:

Explicando as estruturas Band e Step.

A ideia dessas estruturas é melhorar a manutenibilidade do código com a evolução do projeto. Essas estruturas foram projetadas para ofderecerem um comportamento distinto para cada banda. Ou seja, para sintonizar estações de FM, por exemplo, só é necessário 1 dígito para casa decimal, passos entre 50KHz e 500KHz são mais apropriados que passos muito pequenos como 1KHz usado em SW. Já em AM, passos entre 500Hz e 50KHz, bem como 2 dígitos após a vírgula, creio que funcionam bem . Em SW, é desejável uma precisão melhor, então o passo entre 100Hz e 50KHz, bem como duas casas após a vírgula pode ser mais interessante. Então vamos a tabela para implementar comportamentos distintos por banda:

O código em C que implementa a tabela pode ser visto a seguir:

// Band database. You can change the band ranges if you need.

// The unit of frequency here is 0.01Hz (1/100 Hz). See Etherkit Library at https://github.com/etherkit/Si5351Arduino

Band band[] = {

{"MW ", 50000000LLU, 170000000LLU, 45500000LU, " KHz", 100000.0f, 2, 3, 6, 5},

{"SW1 ", 170000000LLU, 1000000000LLU, 45500000LU, " KHz", 100000.0f, 2, 2, 6, 3},

{"SW2 ", 1000000000LLU, 2000000000LLU, 45500000LU, " KHz", 100000.0f, 2, 2, 6, 3},

{"SW3 ", 2000000000LLU, 3000000000LLU, 45500000LU, " KHz", 100000.0f, 2, 2, 6, 3},

{"VHF1 ", 3000000000LLU, 7600000000LLU, 45500000LU, " KHz", 100000.0f, 2, 2, 7, 3},

{"FM ", 7600000000LLU, 10800000000LLU, 1075000000LLU, " MHz", 100000000.0f, 1, 6, 8, 7},

{"AIR ", 10800000000LLU, 13500000000LLU, 1075000000LLU, " MHz", 100000000.0f, 2, 2, 7, 5},

{"VFH2 ", 13500000000LLU, 16000000000LLU, 1075000000LLU, " MHz", 100000000.0f, 2, 2, 7, 5}};

Ainda em relação ao código anterior, lembre-se que band[] é do tipo definido Band, que é declarado como se segue:

typedef struct

{

char \*name;

uint64\_t minFreq; // Min. frequency value for the band (unit 0.01Hz)

uint64\_t maxFreq; // Max. frequency value for the band (unit 0.01Hz)

uint64\_t offset;

char \*unitFreq; // MHz or KHz

float divider; // value that will be the divider of current clock (just to present on display)

short decimals; // number of digits after the comma

short initialStepIndex; // Index to the initial step of incrementing

short finalStepIndex; // Index to the final step of incrementing

short starStepIndex; // Index to start step of incrementing

} Band;

Vamos o significado de cada variável do tipo definido Band:

* **char \*name** – É o nome da banda (ponteiro para a “string” que contém o nome da banda);
* **uint64\_t minFreq** - é a menor frequência possível para a banda;
* **uint64\_t maxFreq** - é a maior frequência possível para a banda;
* **uint64\_t offset** - o desconto que tenho que da FI 455KHz para AM e 10.75MHz para FM (e poderia ainda ser diferente para outras bandas que não MW/SW e FM. Creio que isso vai depender da arquitetura do rádio);
* **char \*unitFreq** – É a unidade utilizada para apresentar no display (KHz ou MHz, mas poderia também ser Hz, dependendo do que você queira)
* **float divider** – É o divisor utilizado para acomodar a frequência no Display e alinhar com a unidade apresentada. Isto é, para os si5351 envio a frequência em uma unidade equivalente a 1/100Hz. Para o Display, ajusto esse valor dependendo da Unidade (KHz ou MHZ). O divisor na realidade converte a unidade exigida para o si5351para uma unidade compatível com a banda utilizada (KHz ou MHz);
* **short decimals** – Número de casas decimais que devem ser apresentadas após a vírgula;
* **short initialStepIndex** – Menor passo de incremento/decremento permitido para a banda. O valor armazenado em initialStepIndex é o índice da tabela (array) step[]);
* **short finalStepIndex** – Maior passo de incremento /. Decremento permitido para a banda. . O valor armazenado em finalStepIndexé o índice da tabela (array) step[])
* **short starStepIndex** – E o passo padrão da banda. Quando a banda for selecionada, o passo que será usando está definido em **starStepIndex. Lambrando que o valor armazenado em starStepIndex** é o índice da tabela (array) step[])

Agora voltando a tabela band, observe as informações da primeira banda (índice 0, “MW”):

{"MW ", 50000000LLU, 170000000LLU, 45500000LU, " KHz", 100000.0f, 2, 3, 6, 5}

* **char \*name** – “MW “;
* uint64\_t minFreq - 50000000LLU;
* uint64\_t maxFreq - 170000000LLU;
* uint64\_t offset - 45500000LU;
* char \*unitFreq – " KHz";
* long divider – 100000.0f;
* short decimals – 2
* short initialStepIndex – 3 -> (1KHz) Ver tabela step;
* **short finalStepIndex** – 6 -> (50KHz) Ver tabela step;
* **short starStepIndex** – 5 -> (10KHz) ver tabela step;

// Steps database. You can change the Steps and numbers of steps here if you need.

Step step[] = {

{"10Hz ", 1000}, // index 0

{"100Hz ", 10000}, // 1

{"500Hz ", 50000}, // 2

{"1KHz ", 100000}, // 3

{"5KHz ", 500000}, // 4

{"10KHz ",1000000}, // 5

{"50KHz ", 5000000}, // 6

{"100KHz", 10000000}, // 7

{"500KHz", 50000000}}; // 8

Vamos agora ao uso dessas tabelas/estruturas no programa:

Quando eu altero a Banda, faço o seguinte:

if (digitalRead(BUTTON\_BAND) == HIGH && (millis() - elapsedButton) > MIN\_ELAPSED\_TIME)

{

currentBand = (currentBand < lastBand) ? (currentBand + 1) : 0; // Is the last band? If so, go to the first band (AM). Else. Else, next band.

vfoFreq = band[currentBand].minFreq;

currentStep = band[currentBand].starStepIndex;

isFreqChanged = true;

elapsedButton = millis();

}

Note o código

currentStep = band[currentBand].starStepIndex;

o código acima faz o step na informação guardada em startStepIndex.

Mudança do Step:

else if (digitalRead(BUTTON\_STEP) == HIGH && (millis() - elapsedButton) > MIN\_ELAPSED\_TIME)

{

if (currentClock == 0) // Is VFO

currentStep = (currentStep < band[currentBand].finalStepIndex) ? (currentStep + 1) : band[currentBand].initialStepIndex; // Increment the step or go back to the first

else // Is BFO

currentStep = (currentStep < lastStepBFO) ? (currentStep + 1) : 0;

isFreqChanged = false;

clearDisplay = true;

elapsedButton = millis();

}

Na mudança de step, verifico os limites permitidos para o step. Onserve o código

currentStep = (currentStep < band[currentBand].finalStepIndex) ? (currentStep + 1) : band[currentBand].initialStepIndex;

Neste código, testo se o passo corrente (currentStep) é menor que o máximo permitido para a banda “**band[currentBand].finalStepIndex**”. Caso afirmativo, incremento 1 para ir para o próximo passo (currentStep + 1); caso contrário, vou para o primeiro passo permitido para a banda “**band[currentBand].initialStepIndex”.**

Na função Display, também uso esta tabela. Observe:

void displayDial()

{

double vfo = (double)vfoFreq / band[currentBand].divider;

double bfo = (double)bfoFreq / 100000.0f;

String mainFreq;

String secoundFreq;

String staticFreq;

String dinamicFreq;

// Change the display behaviour depending on who is controlled, BFO or BFO.

if (currentClock == 0)

{ // If the encoder is controlling the VFO

mainFreq = String(vfo, band[currentBand].decimals);

mainFreq.concat(band[currentBand].unitFreq);

secoundFreq = String(bfo, 2);

staticFreq = "BFO";

dinamicFreq = "VFO";

}

else // encoder is controlling the VFO

{

mainFreq = String(bfo, 2);

secoundFreq = String(vfo, band[currentBand].decimals);

secoundFreq.concat(band[currentBand].unitFreq);

staticFreq = "VFO";

dinamicFreq = "BFO";

}

**...**

No código

double vfo = (double)vfoFreq / band[currentBand].divider;

faço a divisão pelo valor de band[currentBand].divider.

Note também que se for o VFO a principal informação, uso o número de casas decimais adequadas para a banda. Veja o código:

mainFreq = String(vfo, band[currentBand].decimals);

e também apresento a unidade de frequência selecionada para a banda. Veja:

mainFreq.concat(band[currentBand].unitFreq);

obs: concat é um método da classe string que junta duas strings. Dessa forma a frequência e a unidade ficarão na mesma string.

Bom, tentei apresentar os elementos do uso das tabelas band[] e step[]. Sei que devo ter deixado a solução um pouco mais complexa de entender no início. PORÉM imagine o seguinte:

1. Se eu tivesse que implementar o comportamento de cada banda com IF e ELSE ou CASE no código. Como ficaria esse código? Mais legível? Para uma ou duas Bandas provavelmente sim, para várias bandas, certamente não.
2. Se eu precisar retirar uma banda do meu sistema é só excluir a linha da tabela referente a banda. Não preciso fazer mais nada no código. Só compilar e aquela banda não fará mais parte do sistema. E como seria se você tivesse implementado a banda no código? Procuraria o IF, ELSE ou CASE relacionados e removeria o código. Mas será que isso não é mais trabalhoso e sujeito a incluir erros no código?
3. Na mesma analogia com o item anterior, se eu precisar adicionar mais uma banda (subdividindo as outras), tudo que é preciso fazer é adicionar mais uma linha com as informações da nova banda na tabela band[]. Como seria usando codificação com IF, ELSE ou CASE? Mais fácil?

Por fim, Marcos, essa foi a minha abstração para o problema atual do VFO / BFO. Não quero promover a ideia de que é uma excelente solução ou ainda que é melhor que outras. Somente estruturei o código para ficar mais adaptável à situações que poderão chegar.