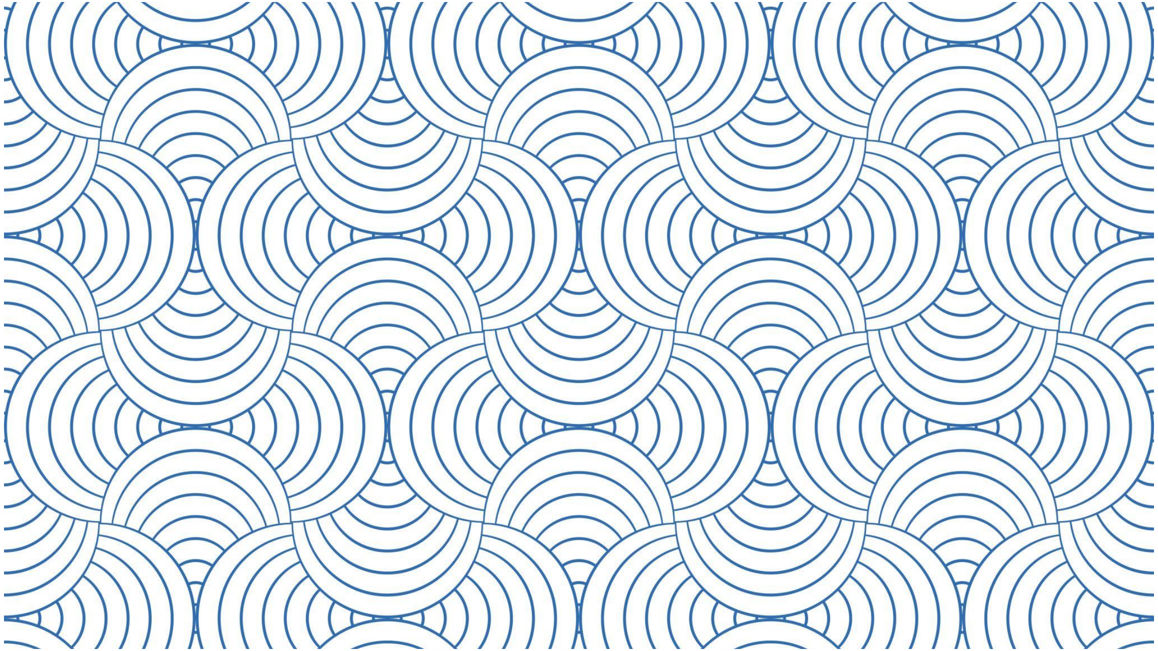
GM2 Proposal: Lao Bytecode



Oisín Conlon

Diya Thomas

Oliver Lee

Laos

Population:

7.5 million

Economy: Primarily agricultural

60–70% of people work in farming

Language: Lao (ລາວ)

Limited access to tech tools in local script

Rural infrastructure challenges: low internet penetration, low literacy in tech

Literacy rate: 85%

Digital literacy even lower

THE MAKER BOX

Who are Makerbox?

Makerbox is a company based in Laos who aim to solve a range of problems facing people in Laos with technology and engineering solutions.

Given the rural demographic in Laos, one of the most important ways to do this is to pioneer new solutions for the improvement of the country’s agriculture sector

What is our problem?

1

In order to help farmers monitor their farms using cheap automated methods, having screens that can display their language is an essential.

2

In order to this simply, cheaply and in away that is scalable, this must be able to done with microcontrollers.

3

Currently there is no way to do this so that can be run on these small microcontrollers such as Arduinos.

Problems with rendering Lao font on microcontrollers

1

Lao uses a combination of tone marks and vowel signs that are placed above or below a base character.

2

On computers, this complexity is handled by font shaping engines like HarfBuzz and FreeType.

3

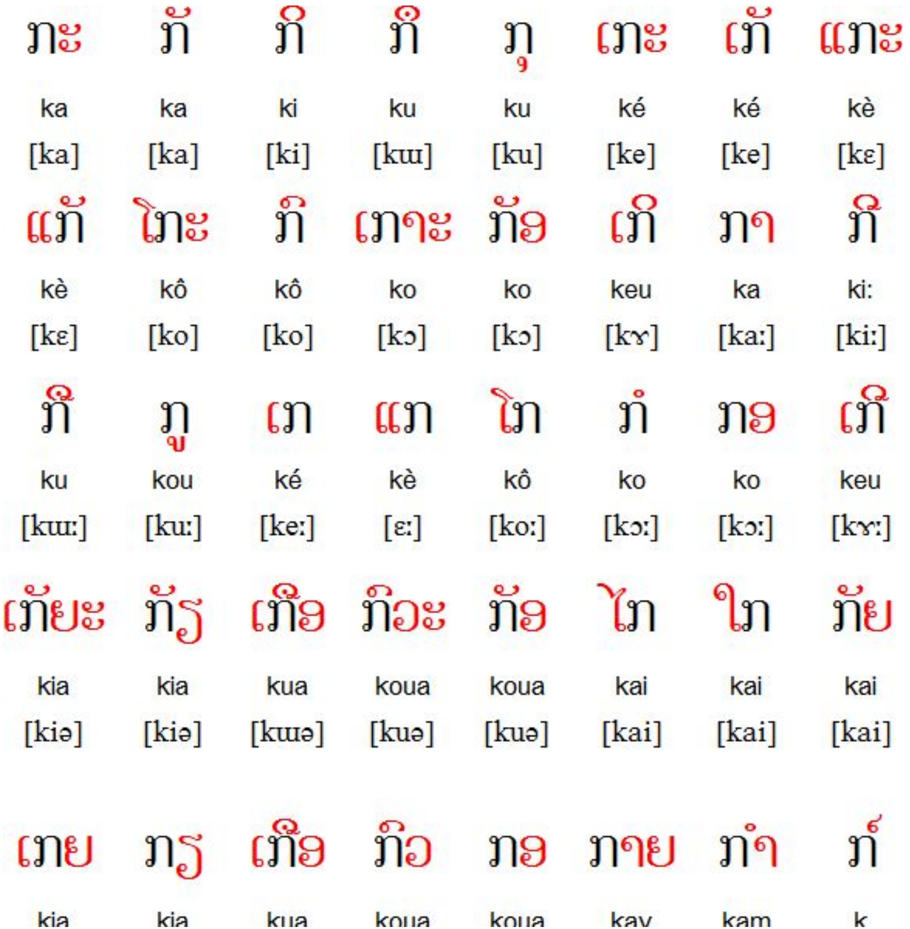
However, microcontrollers lack the memory and processing power to run these font shaping engines.

Solution: Bitmap fonts store the characters as pixel arrays and are easy to load and render making them ideal for embedded system applications!

Our approach

- To do this we are using Bytecode which allows us to encode letters in a far less memory intensive manner than conventional vector fonts.
- This requires us to create interpretable Lao characters from scratch.
- While success has been had with the consonants, the structure of Lao means that the implementation of vowels has proven to be complex.

- Additionally, no resolution scaling is available without distortion of the font. Therefore to support multiple font sizes many versions of the bitmap must be created.



ກະ	ກັ	ກິ	ກຶ	ກຸ	ເກະ	ເກັ	ແກະ
ka	ka	ki	ku	ku	ké	ké	kè
[ka]	[ka]	[ki]	[kɯ]	[ku]	[ke]	[ke]	[kɛ]
ແກັ	ໂກະ	ກົ	ເກາະ	ກັອ	ເກິ	ກາ	ກີ
kè	kô	kô	ko	ko	keu	ka	ki:
[kɛ]	[ko]	[ko]	[kɔ]	[kɔ]	[kɤ]	[ka:]	[ki:]
ກື	ກູ	ເກ	ແກ	ໂກ	ກໍ	ກອ	ເກີ
ku	kou	ké	kè	kô	ko	ko	keu
[kɯ:]	[ku:]	[ke:]	[ɛ:]	[ko:]	[kɔ:]	[kɔ:]	[kɤ:]
ເກັຍະ	ກັຽ	ເກືອ	ກົວະ	ກັອ	ໄກ	ໃກ	ກັຍ
kia	kia	kua	koua	koua	kai	kai	kai
[kiə]	[kiə]	[kɯə]	[kuə]	[kuə]	[kai]	[kai]	[kai]
ເກຍ	ກຽ	ເກືອ	ກົວ	ກອ	ກາຍ	ກຳ	ກ໌
kia	kia	kua	koua	koua	kay	kam	k

Multidisciplinary Project

- This will require us to adjust how the letters are encoded and how they are displayed on the screen to find an optimum.
- Marrying the software side with the hardware side is the key to our success

The hardware side

The software side

Three solutions to explore…

01 Continuing MakerBox's current solution

- The current solution involves pre-rendering the most common Lao syllables (combinations of base letters and marks) into individual bitmap glyphs.
- While this eliminates the need for shaping or mark positioning, this doesn't scale well as supporting every valid Lao combination would require hundreds of glyphs.
- A specific keyboard would need to be designed for this solution (which would pose a problem in fitting all the combinations onto the keyboard space.)

02 Rule-Based Rendering

- A second solution would be to develop a rule based program which uses a simple set of rules to decide where to place tone marks. Eg. If the incoming text includes a base consonant followed by a vowel, the program would detect this and then place the vowel a fixed offset above or below the consonant.
- This solution would be compatible with a standard Lao keyboard and Unicode.
- It would not require as many bitmapped glyphs - the individual characters would suffice
- However, there may be problems with the overlapping characters due to inadequate offsets and multiple stacked marks could be a problem.

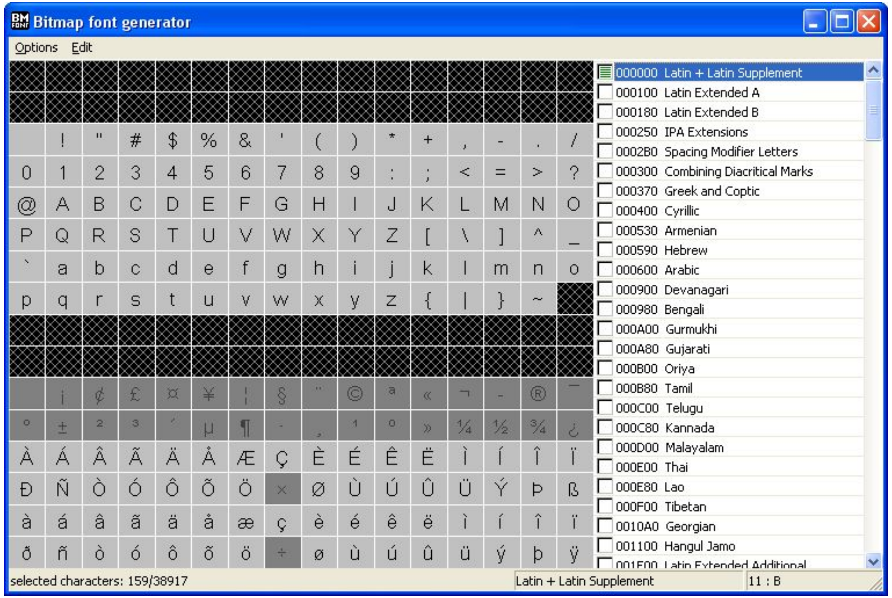
03 Pre-processing of the text

- The text could be shaped and preprocessed by the computer sending the message (which has higher compute power). This could then be converted into a set of specific drawing instructions to be sent to the microcontroller. (Eg. Exact offsets after considering each character's height.)

Creating the Bitmap

We plan on using open source software BMFont to help generate the bitmap for the characters.

Alternatively, we can use Google fonts (https://fonts.google.com) and export a vector font (TTF) file for the Lao language.



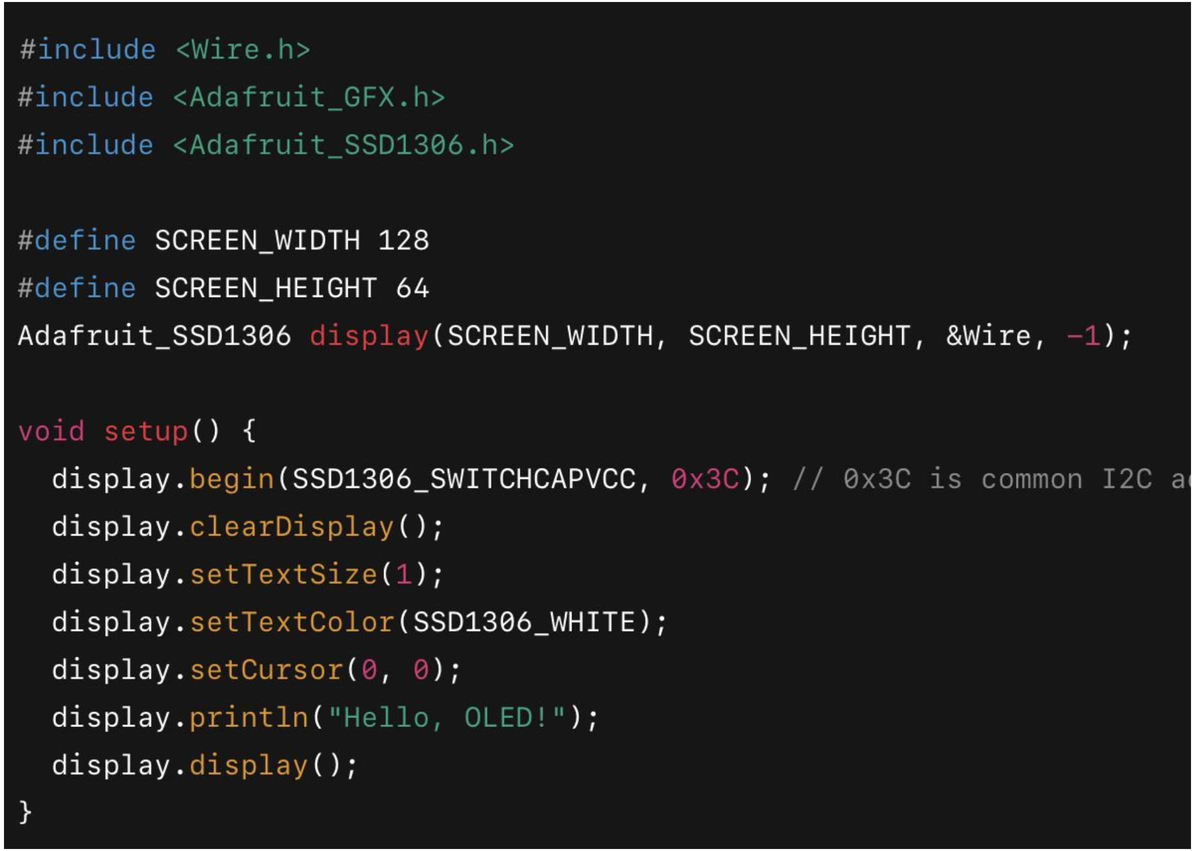
	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

- ▤ 000000 Latin + Latin Supplement
- □ 000100 Latin Extended A
- □ 000180 Latin Extended B
- □ 000250 IPA Extensions
- □ 0002B0 Spacing Modifier Letters
- □ 000300 Combining Diacritical Marks
- □ 000370 Greek and Coptic
- □ 000400 Cyrillic
- □ 000530 Armenian
- □ 000590 Hebrew
- □ 000600 Arabic
- □ 000900 Devanagari
- □ 000980 Bengali
- □ 000A00 Gurmukhi
- □ 000A80 Gujarati
- □ 000B00 Oriya
- □ 000B80 Tamil
- □ 000C00 Telugu
- □ 000C80 Kannada
- □ 000D00 Malayalam
- □ 000E00 Thai
- □ 000E80 Lao
- □ 000F00 Tibetan
- □ 0010A0 Georgian
- □ 001100 Hangul Jamo
- □ 001E00 Latin Extended Additional

selected characters: 159/38917 | Latin + Latin Supplement | 11 : B

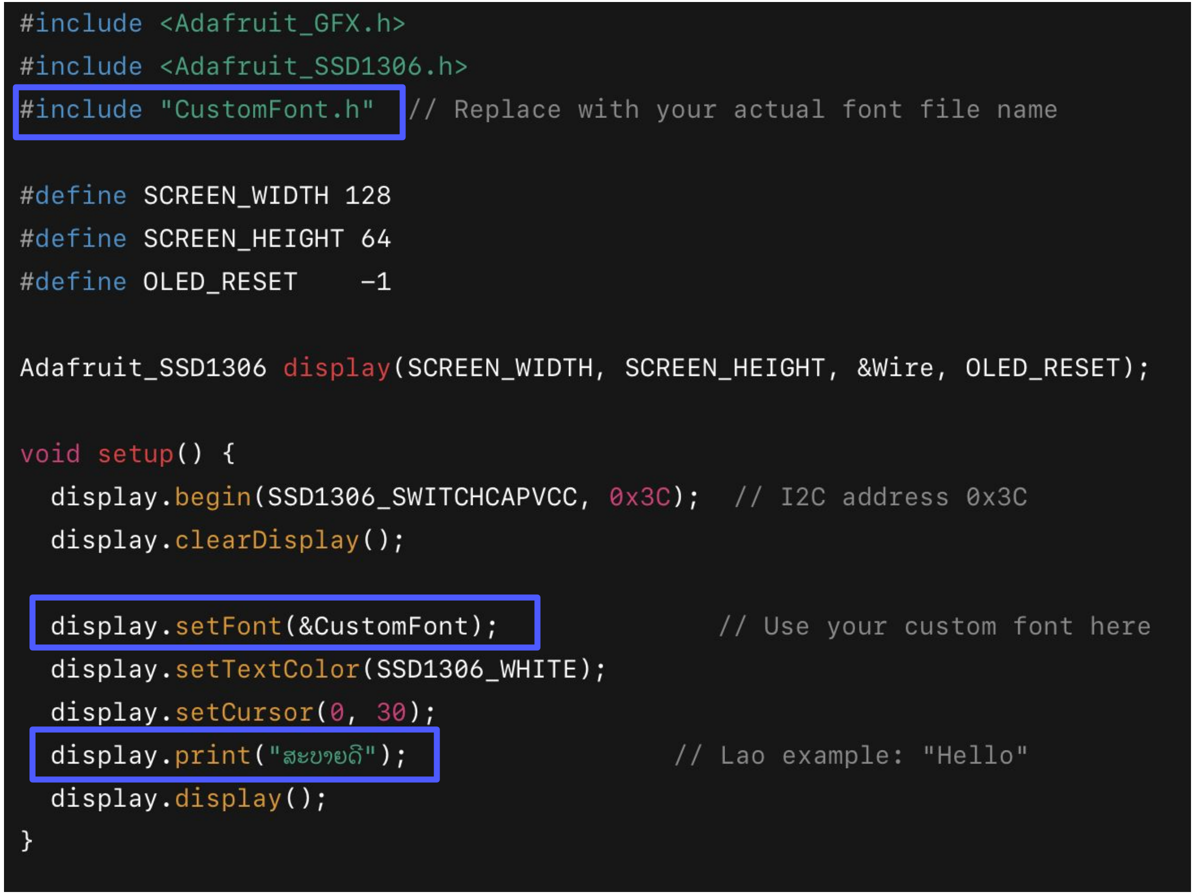
Displaying text in English

Uses the **Adafruit_SSD1306** library display.println function



```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

void setup() {
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C); // 0x3C is common I2C a
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  display.println("Hello, OLED!");
  display.display();
}
```

Displaying a custom font



```
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include "CustomFont.h"  // Replace with your actual font file name

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET    -1

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

void setup() {
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);  // I2C address 0x3C
  display.clearDisplay();

  display.setFont(&CustomFont);              // Use your custom font here
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 30);
  display.print("ສະບາຍດີ");                 // Lao example: "Hello"
  display.display();
}
```

To use a custom font with the Adafruit SSD1306 and Adafruit GFX libraries, you need to convert a vector font (TTF or OTF) font into a GFX-compatible.h file using the **Adafruit GFX Font Converter**

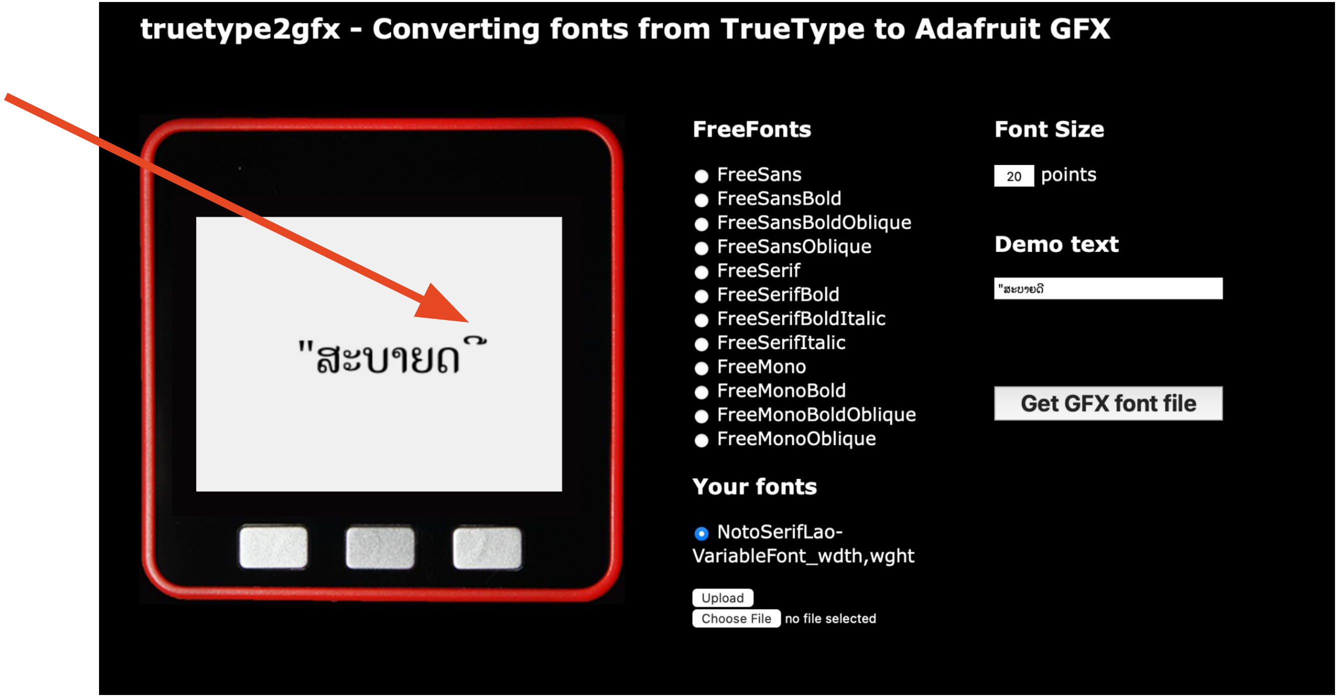
BUT we run into our main problem..

Should read as:

**"ສະບາຍ
ດີ**

With the vowel positioned above the consonants.

They are different Unicode characters (`U+0E94` and `U+0EB5`) so they are rendered as two separate glyphs.



truetype2gfx - Converting fonts from TrueType to Adafruit GFX

"ສະບາຍດ ີ

FreeFonts

○ FreeSans
○ FreeSansBold
○ FreeSansBoldOblique
○ FreeSansOblique
○ FreeSerif
○ FreeSerifBold
○ FreeSerifBoldItalic
○ FreeSerifItalic
○ FreeMono
○ FreeMonoBold
○ FreeMonoBoldOblique
○ FreeMonoOblique

Your fonts

● NotoSerifLao-VariableFont_wdth,wght

Upload
Choose File no file selected

Font Size

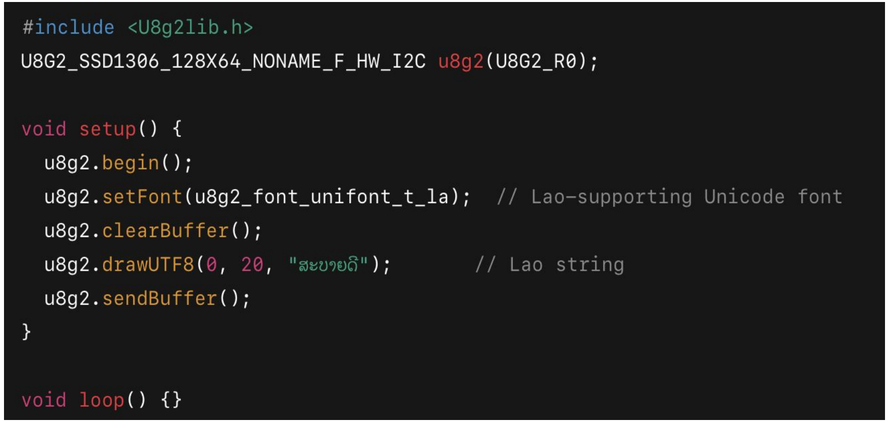
20 points

Demo text

"ສະບາຍດີ

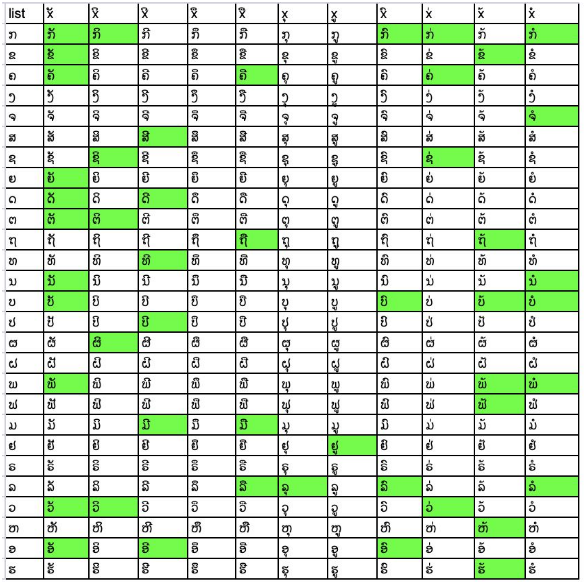
Alternative

The **U8g2 library** with the **Unifont** font does a decent job for fixed-size Lao glyphs. It doesn't shape them, but it includes precomposed glyphs (like ດີ as one block), which helps avoid smearing.



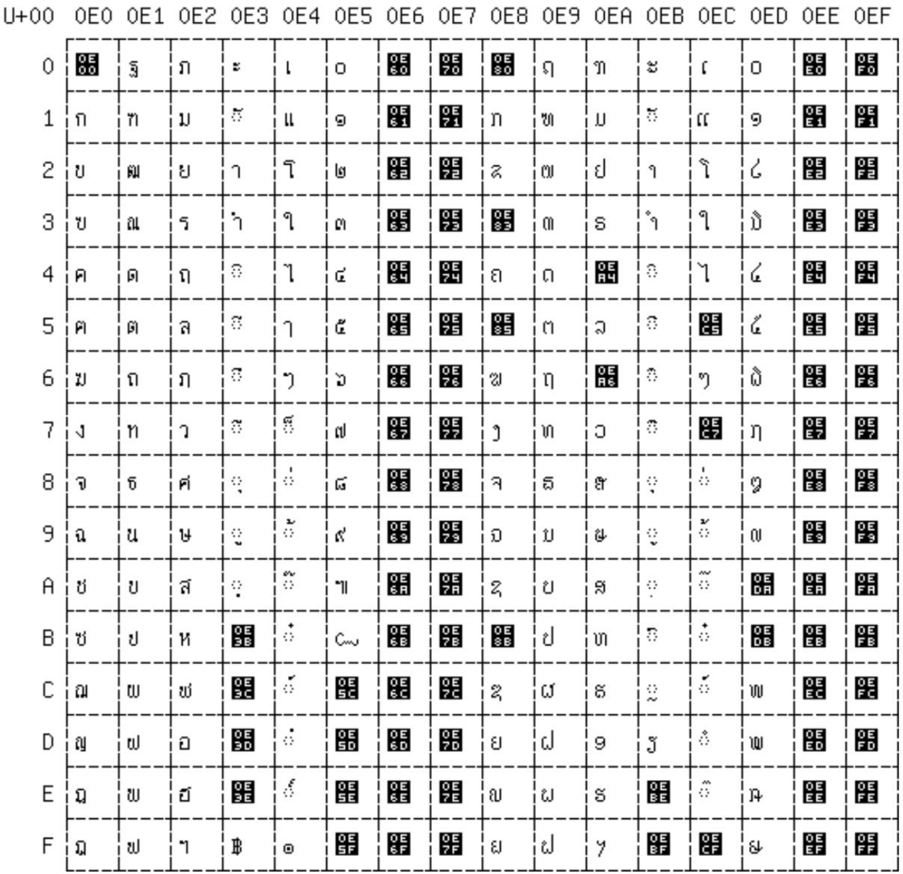
```
#include <U8g2lib.h>
U8G2_SSD1306_128X64_NONAME_F_HW_I2C u8g2(U8G2_R0);

void setup() {
  u8g2.begin();
  u8g2.setFont(u8g2_font_unifont_t_la);  // Lao-supporting Unicode font
  u8g2.clearBuffer();
  u8g2.drawUTF8(0, 20, "ສະບາຍດີ");       // Lao string
  u8g2.sendBuffer();
}

void loop() {}
```

BUT only some precomposed glyphs



list	xັ	xິ	xີ	xຶ	xື	xຸ	xູ	xົ	x່	x້	xໍ
ກ	ກັ	ກິ	ກີ	ກຶ	ກື	ກຸ	ກູ	ກົ	ກ່	ກ້	ກໍ
ຂ	ຂັ	ຂິ	ຂີ	ຂຶ	ຂື	ຂຸ	ຂູ	ຂົ	ຂ່	ຂ້	ຂໍ
ຄ	ຄັ	ຄິ	ຄີ	ຄຶ	ຄື	ຄຸ	ຄູ	ຄົ	ຄ່	ຄ້	ຄໍ
ງ	ງັ	ງິ	ງີ	ງຶ	ງື	ງຸ	ງູ	ງົ	ງ່	ງ້	ງໍ
ຈ	ຈັ	ຈິ	ຈີ	ຈຶ	ຈື	ຈຸ	ຈູ	ຈົ	ຈ່	ຈ້	ຈໍ
ສ	ສັ	ສິ	ສີ	ສຶ	ສື	ສຸ	ສູ	ສົ	ສ່	ສ້	ສໍ
ຊ	ຊັ	ຊິ	ຊີ	ຊຶ	ຊື	ຊຸ	ຊູ	ຊົ	ຊ່	ຊ້	ຊໍ
ຍ	ຍັ	ຍິ	ຍີ	ຍຶ	ຍື	ຍຸ	ຍູ	ຍົ	ຍ່	ຍ້	ຍໍ
ດ	ດັ	ດິ	ດີ	ດຶ	ດື	ດຸ	ດູ	ດົ	ດ່	ດ້	ດໍ
ຕ	ຕັ	ຕິ	ຕີ	ຕຶ	ຕື	ຕຸ	ຕູ	ຕົ	ຕ່	ຕ້	ຕໍ
ຖ	ຖັ	ຖິ	ຖີ	ຖຶ	ຖື	ຖຸ	ຖູ	ຖົ	ຖ່	ຖ້	ຖໍ
ທ	ທັ	ທິ	ທີ	ທຶ	ທື	ທຸ	ທູ	ທົ	ທ່	ທ້	ທໍ
ນ	ນັ	ນິ	ນີ	ນຶ	ນື	ນຸ	ນູ	ນົ	ນ່	ນ້	ນໍ
ບ	ບັ	ບິ	ບີ	ບຶ	ບື	ບຸ	ບູ	ບົ	ບ່	ບ້	ບໍ
ປ	ປັ	ປິ	ປີ	ປຶ	ປື	ປຸ	ປູ	ປົ	ປ່	ປ້	ປໍ
ຜ	ຜັ	ຜິ	ຜີ	ຜຶ	ຜື	ຜຸ	ຜູ	ຜົ	ຜ່	ຜ້	ຜໍ
ຝ	ຝັ	ຝິ	ຝີ	ຝຶ	ຝື	ຝຸ	ຝູ	ຝົ	ຝ່	ຝ້	ຝໍ
ພ	ພັ	ພິ	ພີ	ພຶ	ພື	ພຸ	ພູ	ພົ	ພ່	ພ້	ພໍ
ຟ	ຟັ	ຟິ	ຟີ	ຟຶ	ຟື	ຟຸ	ຟູ	ຟົ	ຟ່	ຟ້	ຟໍ
ມ	ມັ	ມິ	ມີ	ມຶ	ມື	ມຸ	ມູ	ມົ	ມ່	ມ້	ມໍ
ຢ	ຢັ	ຢິ	ຢີ	ຢຶ	ຢື	ຢຸ	ຢູ	ຢົ	ຢ່	ຢ້	ຢໍ
ຣ	ຣັ	ຣິ	ຣີ	ຣຶ	ຣື	ຣຸ	ຣູ	ຣົ	ຣ່	ຣ້	ຣໍ
ລ	ລັ	ລິ	ລີ	ລຶ	ລື	ລຸ	ລູ	ລົ	ລ່	ລ້	ລໍ
ວ	ວັ	ວິ	ວີ	ວຶ	ວື	ວຸ	ວູ	ວົ	ວ່	ວ້	ວໍ
ຫ	ຫັ	ຫິ	ຫີ	ຫຶ	ຫື	ຫຸ	ຫູ	ຫົ	ຫ່	ຫ້	ຫໍ
ອ	ອັ	ອິ	ອີ	ອຶ	ອື	ອຸ	ອູ	ອົ	ອ່	ອ້	ອໍ
ຮ	ຮັ	ຮິ	ຮີ	ຮຶ	ຮື	ຮຸ	ຮູ	ຮົ	ຮ່	ຮ້	ຮໍ

Green are bitmaps created by Bill (15pt)



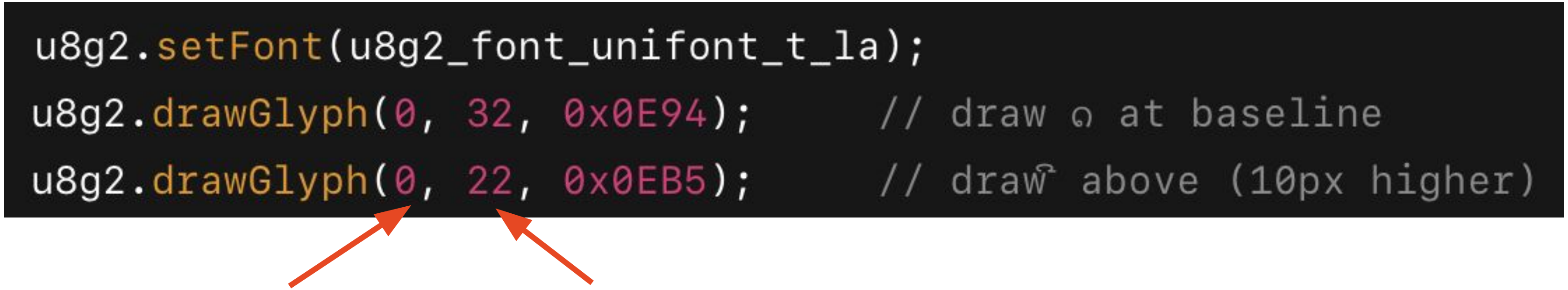
U+00	0E0	0E1	0E2	0E3	0E4	0E5	0E6	0E7	0E8	0E9	0EA	0EB	0EC	0ED	0EE	0EF
0	0E00	ฐ	ภ	ะ	เ	๐	0E60	0E70	0E80	ຐ	ຠ	ະ	ເ	໐	0EE0	0EF0
1	ก	ฑ	ม	◌ั	แ	๑	0E61	0E71	ກ	ຑ	ມ	◌ັ	ແ	໑	0EE1	0EF1
2	ข	ฒ	ย	า	โ	๒	0E62	0E72	ຂ	ຒ	ຢ	າ	ໂ	໒	0EE2	0EF2
3	ฃ	ณ	ร	◌ำ	ใ	๓	0E63	0E73	0E83	ຓ	ຣ	◌ຳ	ໃ	໓	0EE3	0EF3
4	ค	ด	ฤ	◌ิ	ไ	๔	0E64	0E74	ຄ	ດ	0EA4	◌ິ	ໄ	໔	0EE4	0EF4
5	ฅ	ต	ล	◌ี	ๅ	๕	0E65	0E75	0E85	ຕ	ລ	◌ີ	0EC5	໕	0EE5	0EF5
6	ฆ	ถ	ฦ	◌ึ	ๆ	๖	0E66	0E76	ຆ	ຖ	0EA6	◌ຶ	ໆ	໖	0EE6	0EF6
7	ง	ท	ว	◌ื	◌็	๗	0E67	0E77	ງ	ທ	ວ	◌ື	0EC7	໗	0EE7	0EF7
8	จ	ธ	ศ	◌ุ	◌่	๘	0E68	0E78	ຈ	ຘ	ຨ	◌ຸ	◌່	໘	0EE8	0EF8
9	ฉ	น	ษ	◌ู	◌้	๙	0E69	0E79	ຉ	ນ	ຩ	◌ູ	◌້	໙	0EE9	0EF9
A	ช	บ	ส	◌ฺ	◌๊	๚	0E6A	0E7A	ຊ	ບ	ສ	◌຺	◌໊	0EDA	0EEA	0EFA
B	ซ	ป	ห	0E3B	◌๋	๛	0E6B	0E7B	0E8B	ປ	ຫ	◌ົ	◌໋	0EDB	0EEB	0EFB
C	ฌ	ผ	ฬ	0E3C	◌์	0E5C	0E6C	0E7C	ຌ	ຜ	ຬ	◌ຼ	◌໌	ໜ	0EEC	0EFC
D	ญ	ฝ	อ	0E3D	◌ํ	0E5D	0E6D	0E7D	ຍ	ຝ	ອ	ຽ	◌ໍ	ໝ	0EED	0EFD
E	ฎ	พ	ฮ	0E3E	◌๎	0E5E	0E6E	0E7E	ຎ	ພ	ຮ	0EBE	◌໎	ໞ	0EEE	0EFE
F	ฏ	ฟ	ฯ	฿	๏	0E5F	0E6F	0E7F	ຏ	ຟ	ຯ	0EBF	0ECF	ໟ	0EEF	0EFF

Table of 16pt Unifont characters

Rules-based Rendering

Implement a function that:

1. Reads UTF-8 Lao text.
2. Splits each syllable into: **Base consonant, Preposed vowel** (before), **Above mark, Below mark, Postposed vowel** (after)
3. Then positions glyphs at custom X/Y offsets based on their function



```
u8g2.setFont(u8g2_font_unifont_t_la);
u8g2.drawGlyph(0, 32, 0x0E94);      // draw ດ at baseline
u8g2.drawGlyph(0, 22, 0x0EB5);      // drawີ above (10px higher)
```

X offset (px)

Y offset (px)

Hardware Choices

In order to demonstrate our MVP, we require a simple & cheap microcontroller and display package.

Considerations:

- Type of display: OLED vs LCD. OLEDs are (fractionally) more expensive but since only individual pixels are illuminated (no backlighting) they have higher resolution and lower energy consumption.
- Size of display: 128x64 vs 128x32. The larger screen will help with displaying the Lao alphabet with its unique vowel placement. Additionally the existing letters created by Bill take the 128x64 format.
- Type of microcontroller: Arduino Nano chosen, it is cheap, compact and includes libraries such as the Adafruit SSD1306 library for displays.

Hardware Costs

Item	Description	Price
Arduino Nano	Simple microcontroller	£4.99 - £21.99
128x64 OLED display	High resolution display	£9.00
Breadboard	For prototyping	£4.78

Parts will be ordered through the engineering department. This means specific components may be chosen through RS Components to minimise lead time.

Project Management Plan

Global Collaboration & Communication

Hybrid team: Laos and Cambridge

Constant internal updates within Cambridge team

Shared progress with Laos team in near-real-time

Strong focus on in-person and tandem collaboration when possible

Key Team Members

Ken Streutker – Project Lead In Laos, Logistics & Coordination

Kabuild (Bill) – Bytecode Expert, Lao Symbol Designer, Strong Written English

Sinthala (Oui) – Head Of TTS, Tech Communication Bridge, Fluent English

Key Team Members

Oliver Lee
Skills: C++ and embedded systems experience
Goals: keen to work on automating any font design as much as possible

Diya Thomas
Skills: Electronics expert
Has done very similar work before with embedded systems
Goals: Apply previously learned skills to a new context

Oisín Conlon
Skills: Microcontroller expert
Linguistic consultant
Goals: Learning how to make the process more memory efficient.

Timeline Overview

1 Week 1: Receive hardware, test characters, learn Lao alphabet, prioritize letters

2 Week 2: Display character strings, generate new letters, support vowels/accents

3 Week 3+: Show key phrases (e.g., 'Add water’), eventually expand to full script

Safety and Risk Awareness

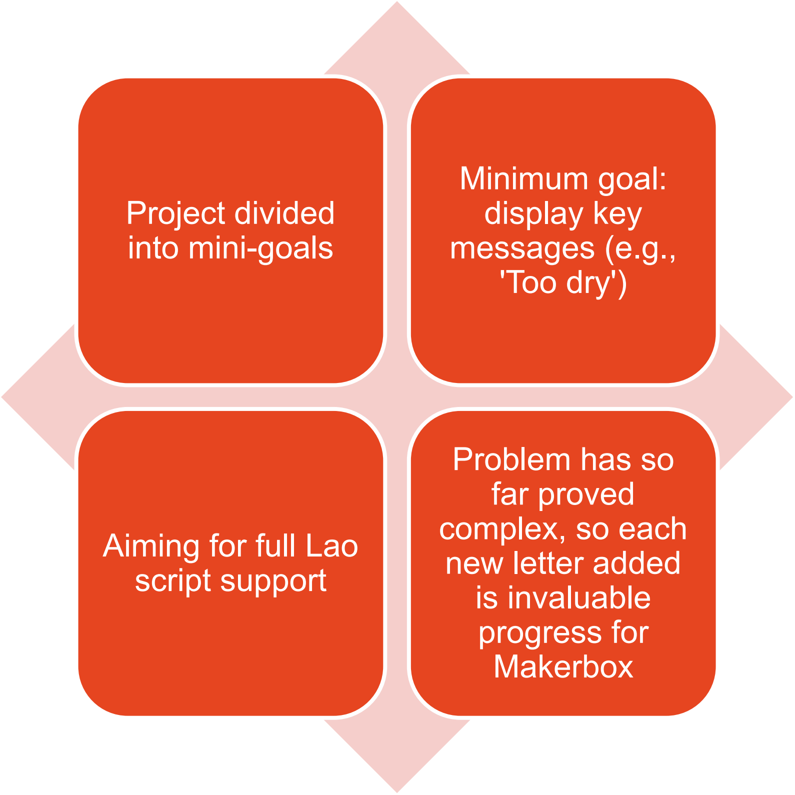
Low-risk project

Standard electronic safety practices

No liquids near devices

Avoiding complacency during testing

Contingency



Project divided into mini-goals

Minimum goal: display key messages (e.g., 'Too dry')

Aiming for full Lao script support

Problem has so far proved complex, so each new letter added is invaluable progress for Makerbox

Quality of Proposal

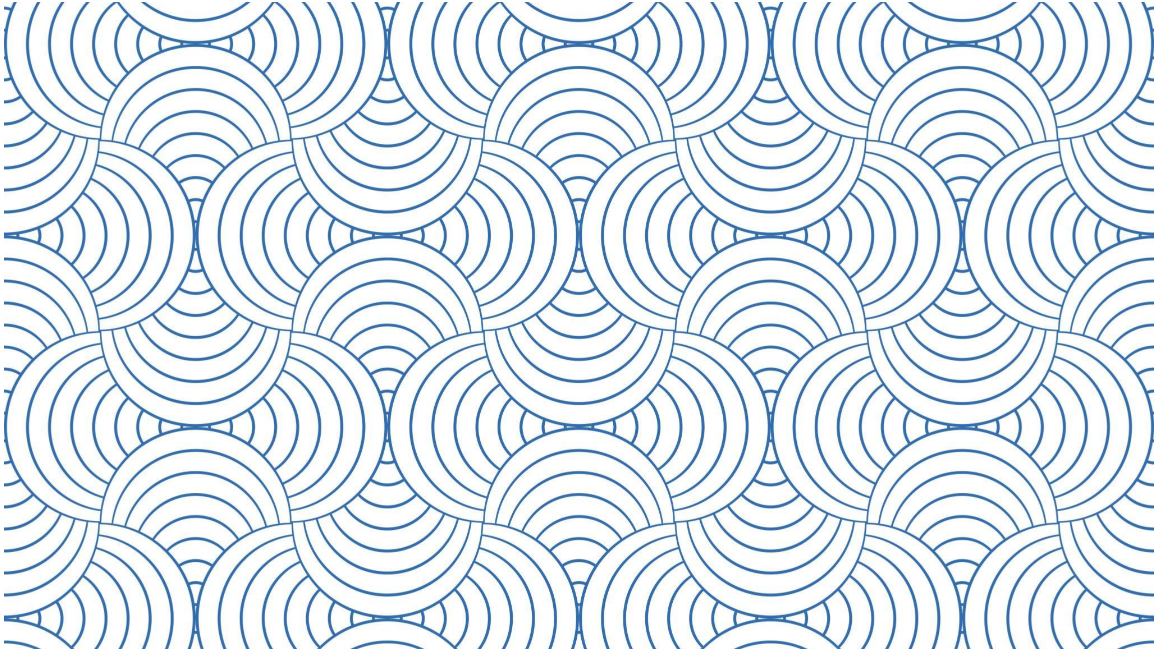
Responding to a genuine need
Collaboratively created solution with team in Laos
Consideration of availability/cost of screens
Successful integration with software ecosystem is vital

Considering the use case and the rural infrastructure challenges (low internet penetration, unreliable energy sources ect.) the MakerBox solution of using bitmapped font on microcontrollers is energy efficient, affordable and impact focused.

Discussions with the team is vital as not only are they experts on the technology, they can also provide us with local context and a perspective that we, as non-natives, may lack.

We may have to consider the availability of chosen display screens and microcontrollers in the Laos market and the additional costs of imports and taxes (perhaps making it compatible with many screen types could be a solution).

Since the majority of our solution will be a software file, it sidesteps the typical challenges of physical systems although successful integration with existing softwares and ecosystems is vital.



Thanks for listening