Środowisko uruchomieniowe dla urządzenia wbudowanego: Lua

Jakub Czapiga, Krystian Życiński

22 stycznia 2019

1 Wprowadzenie

Niniejsza dokumentacja dotyczy projektu z przedmiotu Systemy Wbudowane na Wydziale Informatyki, Elektroniki i Telekomunikacji Akademii Górniczo Hutniczej im. Stanisława Staszica w Krakowie. W dokumencie znajdują się użyte narzędzia, źródła oraz funkcje Lua dostępne na urządzeniu.

2 Użyte narzędzia

2.1 Mikrokontroler

W projekcie został użyty mikrokontroler STM32F746G-DISCO z ekranem dotykowym 480x272. Po dokładne specyfikacje zachęcamy do odwiedzenia strony producenta i dokumentacji produktu.

2.2 Oprogramowanie

Do wygenerowania kodu wykorzystany został STM32CubeMx
Jako środowisko programistyczne został użyty Atollic TRUEStudio
Jądro systemu pochodzi z FreeRTOS.

2.3 Źródła

Jako główne źródło informacji służyła dokumentacja STM32F746G-Disco BSP (Board Support Library). Wykorzystana została także dokumentacja Lua.
Nieocenioną pomoc przyniósł nam dr inż. Robert Brzoza-Woch

3 Działanie platformy uruchomieniowej

Aby użyć interpretera Lua wystarczy:

- Uruchomić urządzenie
- Podpiąć nośnik USB MSC (Pendrive) z programem napisanym w języku Lua. Plik musi być na głównej partycji nośnika oraz nazywać się `script.lua`

4 Działanie wewnętrzne

Przy uruchomieniu urządzenia, w pierwszej kolejności inicjalizowane są wszystkie potrzebne komponenty znajdujące się na urządzeniu. Następnie system rozpoczyna inizjalizację maszyny wirtualnej języka Lua oraz rejestruje wszystkie dostępne moduły (LCS, RTOS, TS). W tym momencie następuje oczekiwanie na dostęp do pliku 1:/script.lua. W przypadku znalezienia pliku (czyli też nośnika) zostaje on załadowany i wykonany przez dostępna maszynę wirtualną.

5 Działanie implementacji API Newlib C

Dostęp do plików został rozwiązany w następujący sposób: Informacje o strukturach są przechowywane wraz z deskryptorem pliku w statucznej globalnej synchronizowanej tablicy. Zapewnia to symulację działania realnego systemu operacyjnego z dostępem do standardowych funkcji STDLIB (strumienie STDIN, STDOUT oraz STDERR są odpowiednio traktowane).

6 Przykłady

Przykład działania znajduje się na repozytoriu projektu w folderze Scripts

7 Przyszłość i kierunek rozwoju platformy

W zamyśle autorów jest stworzenie uniwersalnej biblioteki (BSP-like) pozwalającej na użycie maszyny wirtualnej LUA do prostego rozszerzania funkcjonalności programu bazowego. Może to znaleźć zastosowanie między innymi w personalizacji systemów zautomatyzowanych oraz możliwość tworzenia przez społeczność serwisów do dzielenia się gotowymi funkcjonalnościami.
Proponowanym sposobem działania jest możliwość sterowania maszyną wirtualną za pomocą kolejki rozkazów systemu operacyjnego. Rozważana jest też możliwość uruchomienia modułu jako osobnego zadania, które nie zawłaszcza zasobów i wykonuje się jako częściowo odseparowany wątek.

8 Dokumentacja LUA API

8.1 Moduł RTOS

1. `rtos.yield()` - powrót do schedulera systemowego RTOS. Działanie podobne do `taskYIELD()` w języku C.
2. `rtos.delay(ms)` - powrót do schedulera systemowego RTOS na minimum ms milisekund. Działnie jak `osDelay(ms)` w C.
3. `rtos.get_milis()` - zwraca czas od uruchomienia urządzenia (w milisekundach).
4. `rtos.get_seconds()` - zwraca czas od uruchomienia urządzenia (w sekundach), typ float.

8.2 Moduł TouchScreen

1. `ts.get_status()` - zwraca wartość ostatniego skanu ekranu dotykowego. Struktura wyjściowa
   {touches = [{x = 123, y = 123}, {...} ...], gesture = GESTURE_ID}
   Gesture jest wynikiem wykrywania podstawowych gestów i jest zależne od wsparcia ekranu. Dostępne wartości:
   - GESTURE_NONE - brak gestu
   - GESTURE_MOVE_UP - ruch w górę
   - GESTURE_MOVE_RIGHT - ruch w prawo
   - GESTURE_MOVE_DOWN - ruch w dół
   - GESTURE_MOVE_LEFT - ruch w lewo
   - GESTURE_ZOOM_IN - ruch dwoma palcami do siebie
   - GESTURE_ZOOM_OUT - ruch dwoma palcami od siebie

8.3 Moduł LCD

1. `lcd.get_screen_size()` - zwraca wymiary ekranu jako parę
   { x = x_size, y = y_size }.
2. `lcd.set_text_color(color)` - ustawia kolor rysowanych obiektów. Dostępne kolory w sekcji 3.4 Stałe LCD.
3. `lcd.get_text_color()` - zwraca aktualnie używany kolor.
4. `lcd.set_back_color(color)` - ustawia kolor wypełnienia tła rysowanych obiektów.
5. `lcd.get_back_color()` - zwraca aktualnie używany kolor tła.

6. `lcd.read_pixel(x, y)` - zwraca kolor pixela na pozycji `x,y`. Pozycja dana do funkcji nie może wychodzić poza rozmiar ekranu.
7. `lcd.draw_pixel(x, y, color)` - ustawia pixel na pozycji `x,y` kolorem `color`.
8. `lcd.clear(color)` - czyści bufor kolorem `color`.
9. `lcd.clear_string_line(line)` - czyści daną linię, funkcja używana przy rysowaniu tekstu.
10. `lcd.display_string(line, text)` - wyświetla tekst w danej linii.
11. `lcd.display_string(x, y, text, alignment)` - wyświetla dany tekst na pozycji `x,y` przy wybranym układzie.
12. `lcd.display_char(x, y, c)` - wyświetla dany znak na pozycji `x,y`.
13. `lcd.draw_horizontal_line(x, y, length)` - rysuje poziomą linię długości `length` na pozycji `x,y`.
14. `lcd.draw_vertical_line(x, y, length)` - rysuje pionową linię długości `length` na pozycji `x,y`.
15. `lcd.draw_line(x1, y1, x2, y2)` - rysuje linię między punktem `x1,y1` a punktem `x2,y2`.
16. `lcd.draw_rect(x1, y1, x2, y2)` - rysuje pusty prostokąt między punktami `(x1,y1)` i `x2,y2`.
17. `lcd.draw_circle(x, y, r)` - rysuje pusty okrąg o promieniu `r` ze środkiem w punkcie `x,y`.
18. `lcd.draw_ellipse(x, y, rx, ry)` - rysuje pustą elipsę o promieniach `rx, ry` ze środkiem w punkcie `x,y`.
19. `lcd.fill_rect(x1, y1, x2, y2)` - rysuje wypełniony prostokąt między punktami `(x1,y1)` i `x2,y2`.
20. `lcd.fill_circle(x, y, r)` - rysuje koło o promieniu `r` ze środkiem w punkcie `x,y`.
21. `lcd.fill_ellipse(x, y, rx, ry)` - rysuje wypełnioną elipsę o promieniach `rx, ry` ze środkiem w punkcie `x,y`.
22. `lcd.swap_buffers()` - przenosi obecnie rysowaną klatkę na bufor przedni i wyświetla ją, a poprzednią ustawia jako niewidoczną i gotową do rysowania.
23. `lcd.make_color(r, g, b, a)` - tworzy kolor o parametrach `r,g,b,a`. Parametry muszą być z zakresu `[0.0; 1.0]`.

8.4 Stałe LCD

8.4.1 Kolory

- lcd.COLOR_BLUE
- lcd.COLOR_GREEN
- lcd.COLOR_RED
- lcd.COLOR_CYAN
- lcd.COLOR_MAGENTA
- lcd.COLOR_YELLOW
- lcd.COLOR_LIGHTBLUE
- lcd.COLOR_LIGHTGREEN
- lcd.COLOR_LIGHTRED
- lcd.COLOR_LIGHTCYAN
- lcd.COLOR_LIGHTMAGENTA
- lcd.COLOR_DARKBLUE
- lcd.COLOR_DARKGREEN
- lcd.COLOR_DARKRED
- lcd.COLOR_DARKCYAN
- lcd.COLOR_DARKMAGENTA
- lcd.COLOR_DARKYELLOW
- lcd.COLOR_WHITE
- lcd.COLOR_LIGHTGRAY
- lcd.COLOR_GRAY
- lcd.COLOR_DRKGRAY
- lcd.COLOR_BLACK
- lcd.COLOR_BROWN
- lcd.COLOR_ORANGE
- lcd.COLOR_TRANSPARENT

8.4.2 Pozycjonowanie tekstu

- lcd.CENTER_MODE
- lcd.RIGHT_MODE
- lcd.LEFT_MODE

8.5 Ilość ramek

Ilość ramek używanych do rysowania (N-buffering np. double-buffering dla 2 ramek).

- lcd.LAYERS