# tarea 2 — traductor morse en uefi x64 (nasm) con audio

**estudiante:** Sarah Quesada - 2021046027

**curso y periodo:** sistemas operativos — II 2025

**repositorio:** https://github.com/sarah03qc/morse-uefi.git

**binario:** `BOOTX64.EFI` (se genera en `build/` con make)

---

## 1. introducción

esta tarea implementa una **app uefi x64** escrita en **ensamblador nasm** que: 1) **lee** una oración desde la consola uefi,
2) **traduce** cada carácter a **código morse** y lo imprime en pantalla (utf-16),
3) **reproduce** la secuencia con **pc speaker** por medio del **pit canal 2** (qemu emulado) para que se escuche el audio

**alcance / features listos** - soporte de entrada para `A–Z`, `0–9`, espacio (se muestra como `/`), manejo de **Backspace** - normalización `a..z → A..Z` - control de límites al construir el **outbuf** (utf-16) y terminación `NUL` - audio: **~1 kHz** (pit ch2) con duraciones configurables para `.` y `_` y pausas entre elementos/letra/palabra - todo corre sin edk2: acceso directo a **SystemTable**, **SimpleTextInput**, **SimpleTextOutput** y **BootServices->Stall**

**motivación** - practicar uefi “bare-bones”, convención **ms x64**, llamadas por offset a tablas uefi, manejo de buffers y timing por firmware - además validar el flujo completo de **boot**: preparar esp, copiar `BOOTX64.EFI` y arrancar en qemu/ovmf o usb real

---

## 2. ambiente de desarrollo

- **sistema operativo invitado:** ubuntu (virtualbox) x86_64
- **emulador/firmware:** qemu-system-x86_64 + ovmf
- **ensamblador/enlazador:** `nasm`, `lld-link`
- **utilidades:** `parted`, `dosfstools` (mkfs.vfat), `util-linux` (losetup), `make`
- **hardware real (opcional):** usb en modo uefi (secure boot **off**)

**instalación rapida (ubuntu)** - `sudo apt update && sudo apt install -y nasm lld make qemu-system-x86 ovmf parted dosfstools util-linux`

---

## 3. estructuras de datos usadas y funciones

**constantes/offsets uefi (tablas)** - `OFF_CONIN = 0x30` → `SimpleTextInput*` - `OFF_CONOUT = 0x40` → `SimpleTextOutput*` - `OFF_BOOT_SERVICES = 0x60` → `EFI_BOOT_SERVICES*`

**métodos usados por offset** - `ConOut->OutputString` en `+0x08` - `ConIn->ReadKeyStroke` en `+0x08` - `BootServices->Stall` en `+0xF8`

**puertos / pit / speaker** - `PIT_CMD_PORT = 0x43`, `PIT_CH2_PORT = 0x42`, `SPEAKER_PORT = 0x61` - modo cuadrada ch2: `0xB6` (lobyte/hibyte, mode=3, bin) - divisor elegido ≈ **1000 Hz**

**tiempos (µs)** - `DOT_US = 80000` (≈ 80 ms) - `DASH_US = 8 * DOT_US` - `GAP_INTRA_ELEM_US = 3 * DOT_US` - `GAP_LETTER_EXTRA_US = 6 * DOT_US` - `GAP_WORD_EXTRA_US = 9 * DOT_US`
> intencionalmente “amplificados” para que se oiga claro en emulación, si se quiere estándar **ITU (1/3/1/3/7)** se puede recalibrar

**buffers** - `IN_MAX = 160` (entrada ascii mayúscula) - `OUT_MAX = 2048` (salida **utf-16** con `. _` espacios y `/`)

**tablas de traducción** - `tbl_letters[26]` → punteros a cadenas **utf-16** terminadas en `NUL` para `A..Z` - `tbl_digits[10]` → idem para `0..9` - desconocidos → `?`

**funciones clave** - `morseMain`: obtiene punteros desde `SystemTable`; imprime prompt; bucle de lectura (`ReadKeyStroke`), normaliza a mayúsculas, guarda en `inbuf`; traduce a morse construyendo `outbuf` **utf-16**; muestra resultado y llama a `play_morse_from_utf16` - `play_morse_from_utf16`: recorre la cadena utf-16; `.` -> beep corto; `_` -> beep largo; `' '` pausa de letra, `'/'` pausa de palabra; flag para evitar pausa doble - `beep_for_us(ECX=µs)`: configura PIT ch2, habilita PC speaker (`0x61`), espera `µs` con `stall_us`, apaga speaker - `stall_us`: wrapper a `BootServices->Stall` - `append_space_slash_space`: escribe `/` en **utf-16** dentro del `outbuf`

**abi/convención ms x64** - parámetros: `rcx`, `rdx`, `r8`, `r9`; shadow space 32 bytes; `rsp` alineado a 16 - registros no volátiles: `rbx`, `rbp`, `rdi`, `rsi`, `r12–r15` (se preservan)

---

# 4. instrucciones para compilar y ejecutar

## 4.1 modo imagen (qemu + ovmf) — igual que siempre

```
# flujo completo
make all && make compile && make execute
```

- `all`: crea `img/disk.vhd` (raw), particiona gpt (p1 esp, p2 data), formatea y **monta** p1 en `mnt/`
- `compile`: `nasm -f win64` + `lld-link -subsystem:efi_application -entry:morseMain`, luego copia `build/BOOTX64.EFI` a `mnt/EFI/BOOT/`
- `execute`: lanza qemu con ovmf y **audio** habilitado (pcspk)

### 4.2 modo usb físico (para booteo real)

```
# identificar tu usb en la vm linux (cuidado)
lsblk

# flujo en una linea (BORRA el usb)
make usb-all TARGET_MODE=usb USB_DEVICE=/dev/sdX

# o por pasos
make setup-usb   TARGET_MODE=usb USB_DEVICE=/dev/sdX   #
particiona/forma
make install-usb TARGET_MODE=usb USB_DEVICE=/dev/sdX   # compila y
copia BOOTX64.EFI
```

---

## 5. criterios del enunciado y cómo se cumplen

- **traducción completa a morse**: soporta `A–Z`, `0–9`, espacio como `/`; caracteres fuera del set -> `?`
- **entrada por teclado uefi**: lectura con `SimpleTextInput->ReadKeyStroke`, manejo de **Backspace**
- **salida visible en pantalla**: `SimpleTextOutput->OutputString` sobre cadena **utf-16** construida en `outbuf`
- **audio asociado**: `play_morse_from_utf16` programa **PIT ch2** y conmuta **PC speaker** por `0x61`
- **timing razonable**: constantes `DOT_US`, `DASH_US` y *gaps*; se evitan pausas dobles
- **robustez de buffers**: tamaños `IN_MAX/OUT_MAX`, terminación NUL y chequeos al escribir
- **boot realista**: flujo **imagen** + **qemu/ovmf** y opción **usb físico** con estructura `\EFI\BOOT\BOOTX64.EFI`
- **entregables/documentación**: este markdown incluye ambiente, instrucciones, pruebas, timesheet, autoevaluación y bibliografía

---

## 6. consultas concretas / pruebas que hice

```
entrada: "sos"
salida:  "... ___ ..."
audio: corto-largo con pausas claras

entrada: "hola mundo"
salida:  ".... ___ ._.. ._ / __ __. _._ . _.. ___"
```

---

## 7. actividades realizadas por estudiante (timesheet)

> resumen breve por tarea, fecha y horas

| Fecha | Actividad | Horas |
|---|---|---|

| | | |
|---|---|---|
| 2025-09-07 | repasar enunciado y plan | 0.8 |
| 2025-09-09 | lectura teclado uefi + normalización + buffers | 2.0 |
| 2025-09-10 | tablas morse (utf-16) + outputstring + control de desbordes | 2.2 |
| 2025-09-12 | pit ch2 + pc speaker + tiempos + pausa entre elementos/letras/palabras | 2.5 |
| 2025-09-15 | soporte usb (setup-usb/install-usb), pruebas de booteo | 1.5 |
| 2025-09-15 | realizar documentación | 1 |
| **Total** | | **10** |

# 8. autoevaluación

**estado final**
- entrada por `ReadKeyStroke` con backspace, normalización a mayúsculas
- traducción A–Z y 0–9, espacio como `/`, desconocidos como `?`
- salida **utf-16** con `OutputString`
- audio morse por **pit ch2** + **pc speaker** con pausas coherentes
- build y arranque correctos en **qemu/ovmf** y opción para **usb real**

**problemas encontrados y soluciones**
- **doble qemu** en `execute` (se abrían dos ventanas) -> se dejó **una sola** invocación con audio
- detalle **abi**: preservar no volátiles; para ser ultra estricto, `play_morse_from_utf16` puede salvar/restaurar `rsi` (no rompía, pero es “nice to have”)
- en vm no aparecía el usb -> faltaba extension pack y filtro usb; una vez pasado, `lsblk` mostró `/dev/sdb`

**limitaciones**
- no se incluyen signos de puntuación ni símbolos extra de morse
- tiempos “amplificados”; si se necesita **1/3/1/3/7** exacto, ajustar constantes
- en hardware real el pc speaker puede no existir/estar deshabilitado (el texto si se imprime)

**reporte de commits de git** 2025-09-09 f8f516b UEFI NASM inicial: logramos desplegar ‘Holis’
2025-09-10 f2dacc5 Traductor morse basico y funcional agregado: solo muestra en pantalla (no sound yet)
2025-09-12 1d5ddfe sonido incorporado
2025-09-15 66491d6 Error de duplicacion corregido

**calificación según rubrica (auto)** - traducción morse (correctitud + robustez) — **45/50**
- audio/timing y control de pausas — **18/20**

- boot + flujo de arranque (qemu/usb) — **17/20**
- documentación — **19/20**
- **total** — **99/110**

---

## 9. lecciones aprendidas

- **uefi sin edk2**: entender offsets y punteros a interfaces es muy importante para proyectos tiny
- **utf-16 everywhere**: `OutputString` exige wide, construir outbuf como utf-16 evita dolores
- **timing por firmware**: `BootServices->Stall` da tiempos estables vs busy-waits
- **pit + pc speaker**: programar el canal 2 directo y togglear el puerto `0x61` es simple y didáctico
- **pipelines chiquitos**: primero texto, luego tablas, después audio, validar en pasitos cortos
- **dev-ops de arranque**: imagen gpt + esp, ovmf, qemu, y cuando paso a usb saber que **fat32** + **64.EFI** es la clave

---

## 10. bibliografia

- **uefi spec** (uefi forum), secciones de system table, simple text input/output y boot services (stall)
- **nasm manual**
- **qemu system emulator docs**
- **ovmf (edk ii) docs**
- **itu-r** recomendaciones de temporizacion para código morse (relación 1/3/1/3/7)
- **man-pages linux**: `parted(8)`, `mkfs.vfat(8)`, `losetup(8)`

---

**comandos tipicos** # qemu (imagen): make all && make compile && make execute

```
# usb (borra todo):
lsblk
make usb-all TARGET_MODE=usb USB_DEVICE=/dev/sdX

# ver estado:
make info
```