Université de Liège

Embedded systems project

Final report
Dance Dance Revolution

Simon GARDIER	s192580
Arthur GRAILLET	s182019
Saïd HORMAT-ALLAH	s192779
Camille TRINH	s192024

Année académique 2023-2024

Contents

1 Hardware description

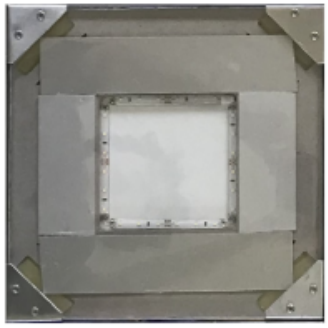
For the complete schematic, refer to `schematics.pdf, page 1`.

1.1 Play pad

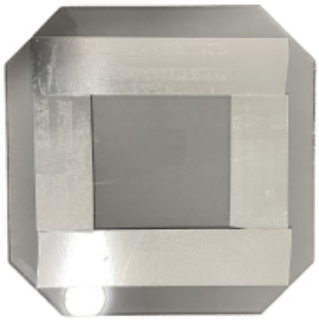
The pad model is highly inspired by this project : Home made Dance Dance Revolution Pad with bar.

1.1.1 Pressure detection system

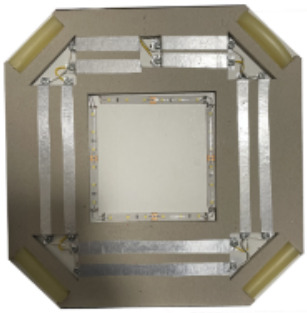
The homemade sensors act similarly to button switches. A plate of polycarbonate is covered with aluminum.
When the player presses a sensor, the top part makes contact with the two lines of aluminum on the part below, (inner square = negative pole, outer square = positive pole) letting the current flows to a pin of the `PIC16f1789`.



Top view of a pad sensor with LED integrated at its center.



Top part, seen from below. Acts as the moving part.



Below part, seen from above. Acts as the fixed part.

Figure 1: Overview of a pad

1.1.2 Light feedback system

1.1.2.1 Description Each homemade pad sensor integrates at its center around 28cm of `SMD3528` 12v DC LEDs strip. The LEDs strips are powered by a 12V DC power supply, each through a separated `IRFZ44N` MOSFET, itself driven by the output of a specific pin of the `PIC16f1789`. When a pressure is detected on the pad, the related LED lights up.

1.1.2.2 Addition of resistors 10k Ohm pull-down resistors are placed on the gates of the `IRFZ44N` to discharge them when no more current is sent from the `PIC16f1789` pin it is connected to, while minimizing current flowing to ground.
1k Ohm current limiter resistors are placed, at the same time, in series between the `PIC16f1789` outputs and the gates of the `IRFZ44N` and in parallel with the 10k resistors. This is done to avoid any damage to the `PIC16f1789` by the gate in-rush current spike created by the transition off → on.

1.2 Main circuit

1.2.1 Power supply and basic setup of the PIC16f1789

The PIC16f1789 is powered using a 6F22 9V Panasonic battery and a voltage regulator as seen during the lab session.
The regulator is composed of an L7805CV transistor and two capacitors of 100µF and 10µF respectively. The regulator is connected to the 9V battery using a diode for protection. A LED is placed in series with RC3 and is used as an ON/OFF indicator.
A connector for a programmer (e.g. Picktit 3) is connected to the MCU to flash the code during the development phase.

1.2.2 Sound system

The sound system has been made as simple as possible, based on : Simple Single Transistor Audio Amplifier Circuit. The circuit is composed of :

- An amplifier circuit made of a 100µF capacitor, a 1k Ohm resistor and a IRFZ44N MOSFET.
- An ABS-222-150-RC 1W 8 Ohm speaker.

1.2.3 Video system

The TV used is a Sony Bravia KDL-32S2000. The TV video entry is connected to the circuit using a AVVCL65 RCA cable. The cable is connected to the system using a PSG01512 RCA connector. The connector is connected to RA0 through a 1000 Ohm resistor and to RC7 through a 470 Ohm resistor. This assembly creates a simple, low tech, yet handy Digital-to-Analog Converter with 4 levels of tension in function of the state of RA0 and RC7.

RA0	RC7	Output voltage	Color
0	0	0	Sync level
0	1	0,33V	Black
1	0	0,67V	Grey
1	1	1V	White

Table 1: Output levels given RA0 and RC7 state

2 Software description

2.1 Frame buffer

To draw the game on the screen, a frame buffer has been implemented. The frame buffer is stored in RAM and is cleared during each vertical synchronization before field one. It is also at that moment that the state of the program is drawn onto the frame buffer. When the program reaches the draw_frame routine, the task two tasks to execute are the generation of the horizontal synchronization signals and the reading/output of the content of the frame buffer on the DAC.

The actual version draw an image of 104×144 pixels. With the last 8 pixels of each line being black. The total amount of RAM used by the frame buffer is : $(104 - 8) \times 144 = 13824$ bits (pixels) $=$ 1728 bytes. Our implementation could theoretically draw an image of 416x576 in terms of speed but the frame could not be stored in the ram, limited to 2048 bytes.

2.2 Tasks list

- τ_1 : Field 1 vertical synchronization
- τ_2 : Field 2 vertical synchronization
- τ_3 : Scan line horizontal synchronization
- τ_4 : Handle user inputs
- τ_5 : Game update
- τ_6 : Menu update
- τ_7 : Music update
- τ_8 : Game draw
- τ_9 : Menu draw
- τ_9 : Draw frame

2.3 Tasks descriptions

- τ_1 : Before the first field, the analog video signal requires to send vertical synchronization signals. See Video signal for more details. The period $T_1 = 40$ ms. Indeed, the PAL standard is based on a frame rate of 25 fps during which the two fields are drawn. Thus, the screen is updated 50 times per second and the time required to drawn a complete image is : 1s / 25 = 0,040s.

  The execution time C_1 is very low since the number of instructions required to send the synchronizations signal is low. This leaves us almost 100% of the task period free.

- τ_2 : Same as τ_1 but for field 2.
- τ_3 : The video lines of the PAL standards must start with an horizontal synchronization signals sequence. The period $T_3 = 64$µs and the execution time C_3 is negligible.
- τ_4 : To make the game feel responsive, we handle the user interactions at each frame. Thus, the period of this task is $T_4 = 40$ms and the execution time $C_4 = 4{,}125$µs.
- τ_5 : At each frame the game is updated : the arrows are moved, the player health and score updated. The period $T_5 = 40$ms, the execution time $C_5 = 104$µs.
- τ_6 : The update of the menu requires no computation most of the time, unless the top arrow is pressed, in which case a new game is prepared to be played. The period $T_6 = 40$ms, the execution time $C_6 = 5$µs.
- τ_7 : The music must be updated frequently to play smoothly. The update is done at each half beat, which last 200ms. The period $T_6 = 200$ms, the execution time $C_6 = 7{,}5$µs.
- τ_8 : We must draw the game at each frame since it is also updated at each frame. The period $T_7 = 40$ms, the execution time $C_7 = 140$µs.
- τ_9 : We must draw the menu at each frame since it is also updated at each frame. The period $T_8 = 40$ms, the execution time $C_8 = 150$µs.
- τ_9 : After each vsync sequence, the PAL standards requires us to output 305 lines. If we want a good resolution, only the output of the pixels on the DAC can be done during this time (along with the necessary horizontal synchronizations). The period $T_8 = 20$ms, the execution time C_8 = almost 20ms.

2.4 Tasks priorities

No specific priority is necessary in our program, only the respect of the PAL signal sequence.

2.5 Interrupt routines

No task require interrupts. This leaves the interrupts free for the any add-on requiring real time response.

2.6 Preemption

No preemption is required in our program. All the tasks can be executed sequentially in the same order each time.

2.7 Program flow

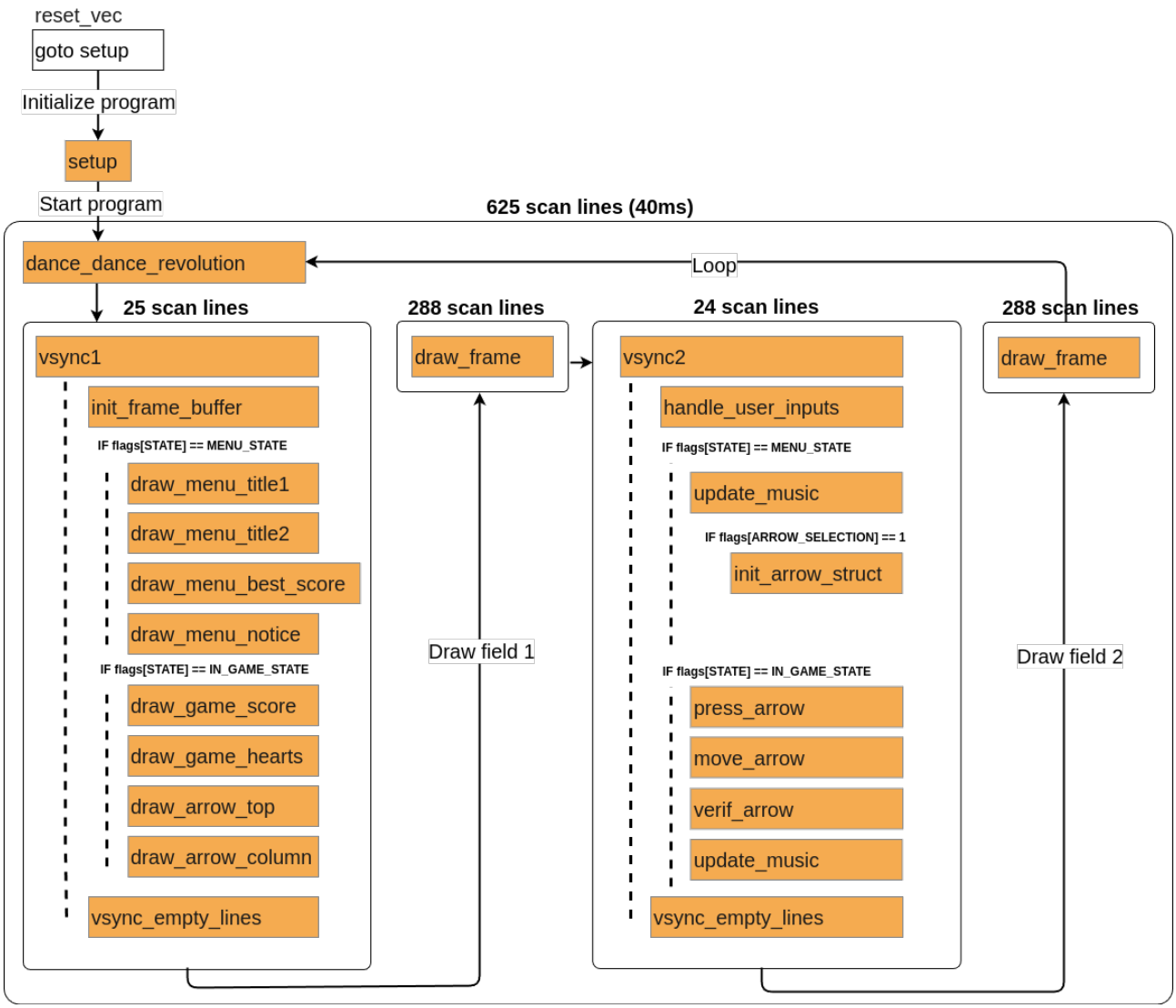


Figure 2: Program flow in the main loop of the program, only "important" functions are shown.

3 Sensors and actuators validation

3.1 Homemade sensor under the pads and reset button

3.1.1 Tests performed

1. Continuity test : test to ensure the circuit works as expected when closed.
2. Detection test : test to turn on an LED when a player pressure is detected in software.

3.1.2 Continuity test

3.1.2.1 Description To test the continuity of the system, we closed the circuit with a red wire to simulate a pressure on the pad and we powered the circuit with the 9V battery. The red wire closing the circuit between the negative and positive pole is in the lower left corner.

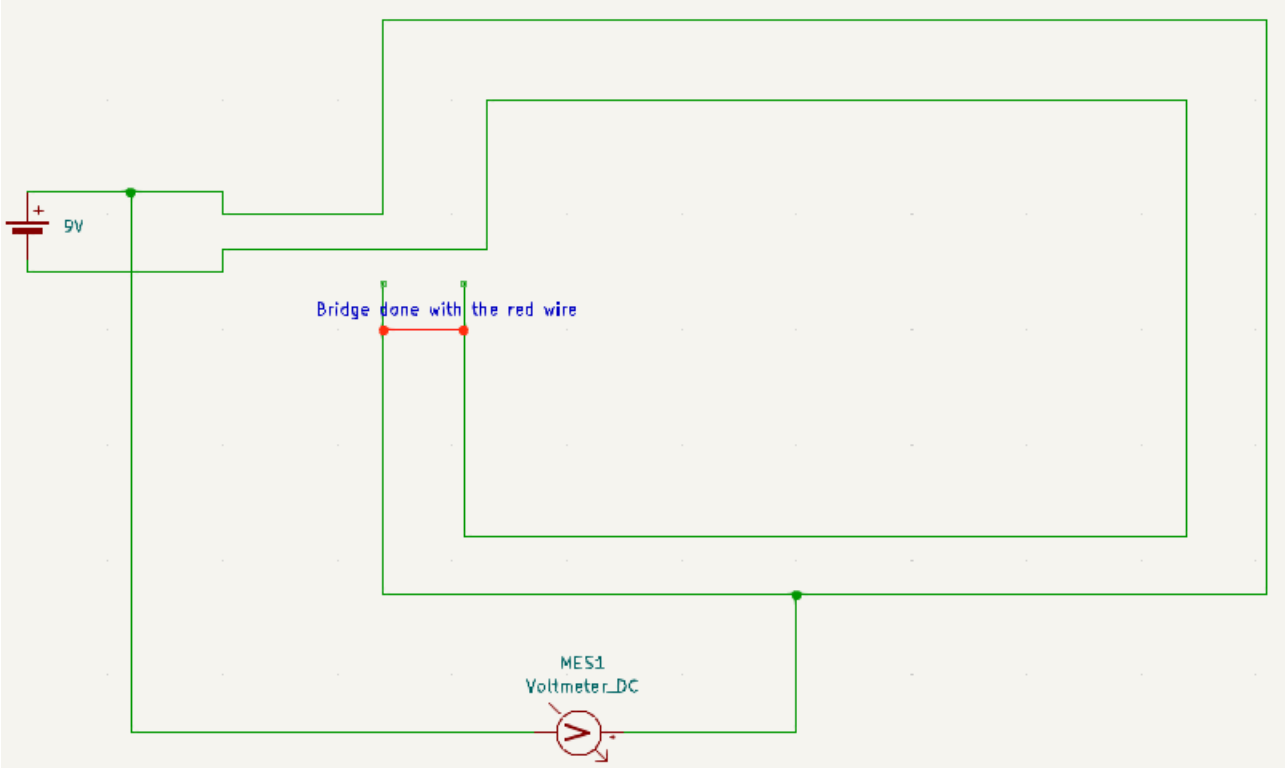


Figure 3: Test with a voltmeter

The multimeter showed a tension of 0V, which is expected since the circuit is closed.
When the red cable is removed, the multimeter shows a tension of 9V. (slightly lower because our battery was not fully charged)

3.1.3 Detection test

3.1.3.1 Description To detect the signal of the pad being pressed, the "Up" pad sensor was connected as shown in `ddr.kicad_sch` and LED of the blinky between RD0 and ground.
A small code is executed on the pic to detect when the pin of the pad is HIGH, in which case it sets RD0 (LED pin) to HIGH.

3.1.3.2 Code Test code related to the LEDs :

```
initialisation:
    ...
    banksel TRISD
    clrf    TRISD       ; Set PORTD as output

    banksel TRISA
    bsf     TRISA, 6    ; Set RA6 as input

    banksel LATD
    clrf    LATD        ; Turn off LED initially
    ...

main_loop:

    ; Read the state of the push button
    banksel PORTA
    btfss   PORTA, 6
    goto    button_not_pressed

    ; If button is pressed, turn on LED
    banksel LATD
    bsf     LATD, 0
    goto    main_loop

button_not_pressed:
    ; If button is not pressed, turn off LED
    banksel LATD
    bcf     LATD, 0
    goto    main_loop
```

3.2 LED

3.2.1 Tests performed

1. Lightning test : test to verify the electrical assembly driving the LEDs.

3.2.2 LED lighting test

To test the LEDs strips, we have build a simple assembly which uses the RD0 pin to drive the LED and the 9V battery to power it.
In this assembly, we output a digital "1" on RD0 using LATD which should light up the led.

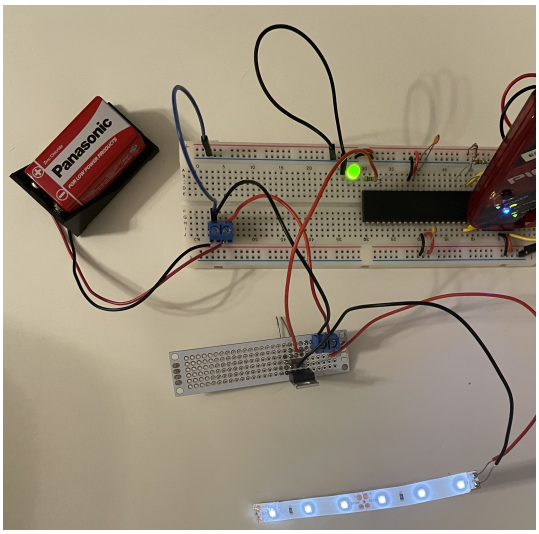
Code related to switching on the LED :

```
initialisation:
    ...
    banksel TRISD
    clrf    TRISD        ; Set PORTD as output

    banksel TRISA
    bsf     TRISA, 6     ; Set RA6 as input

    banksel LATD
    bsf     LATD, 0      ; Turn on LED initially
    ...

main_loop:
    goto    main_loop
```



Legend : The Pickit 3 is used to power the PIC, while the 9V battery power the LED strip through the transistor.

Results: The LED strip lights up.

Electrical assembly:
see `detection.kicad_sch` file.

Figure 4: Lighting of an LED strip

3.3 Speaker

3.3.1 Tests performed

1. Generation of a simple signal using PWM : test to get used to PWM.
2. Powering the speaker using an audio peripheral : test to check the electrical assembly.

3.3.2 Generation of a simple signal using PWM

For this test we used the PWM module to generate a squared signal with a `duty cycle = 50%` (as used for the generation of an audio signal) and a `period = 10µs`.

3.3.2.1 Test code

This code comes from the `PIC16f1789 datasheet, p.241`

```
initialisation:
    BANKSEL PSMC1CON
    MOVLW 0x2         ; set period
    MOVWF PSMC1PRH    ; period high count
    MOVLW 0x7F
    MOVWF PSMC1PRL    ; period low count
    MOVLW 0x01        ; set duty cycle
    MOVWF PSMC1DCH    ; duty cycle high count
    MOVLW 0x3F
    MOVWF PSMC1DCL    ; duty cycle low count
    CLRF PSMC1PHH     ; no phase offset | low count
    CLRF PSMC1PHL     ; phase low count
    MOVLW 0x01        ; PSMC clock=64 MHz
    MOVWF PSMC1CLK    ; clock

    ; output on A, normal polarity
    BSF PSMC1STR0, 0; output pin
    BCF PSMC1POL, 0 ; polarity
    BSF PSMC1OEN, 0 ; output enable control

    ; set time base as source for all events | sync
    BSF PSMC1PRS, 0
    BSF PSMC1PHS, 0
    BSF PSMC1DCS, 0

    ; enable PSMC in Single-Phase Mode
    ; this also loads steering and time buffers
    MOVLW 11000000B
    MOVWF PSMC1CON
    BANKSEL TRISC
    BCF TRISC, 0      ; enable pin driver

main_loop:
    goto main_loop
```

Legend : Signal obtained by connected the oscilloscope between RD0 (PSMC1A) and the ground.

Results : The signal generated is exactly what was expected.

Electrical assembly :
see `ddr.kicad_sch` file (no specific assembly for this test).

Figure 5: PWM output on the oscilloscope.

3.3.3 Powering the speaker using an audio peripheral

To test the electrical assembly the speaker was connected to a vinyl player, resulting in the music being played on the speaker.

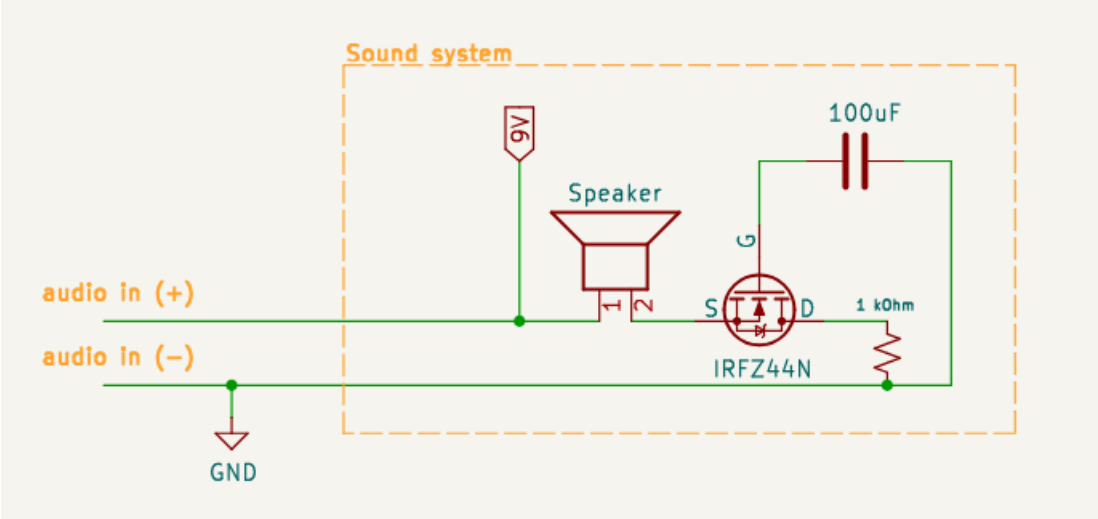


Figure 6: Test assembly

3.4 Video signal

3.4.1 Tests performed

1. Generation of a simple signal using the DAC1 module : test to get used to the generation of analog signals.
2. Generation of a blank image on a screen : test to get used to the generation of PAL signal.

3.4.2 Generation of a simple signal using the DAC1 module

The signal chosen to be generated by the DAC was a simple PAL signal displaying a white screen. The signal is composed of :

- A field-synchronizing sequence for vertical synchronization.
- A field composed of 305 scan-lines themselves composed of :
  - a line-synchronizing sequence for horizontal synchronization
  - a video line

Here is what the field-synchronizing and the line-synchronization sequences should look like according to the ITU.

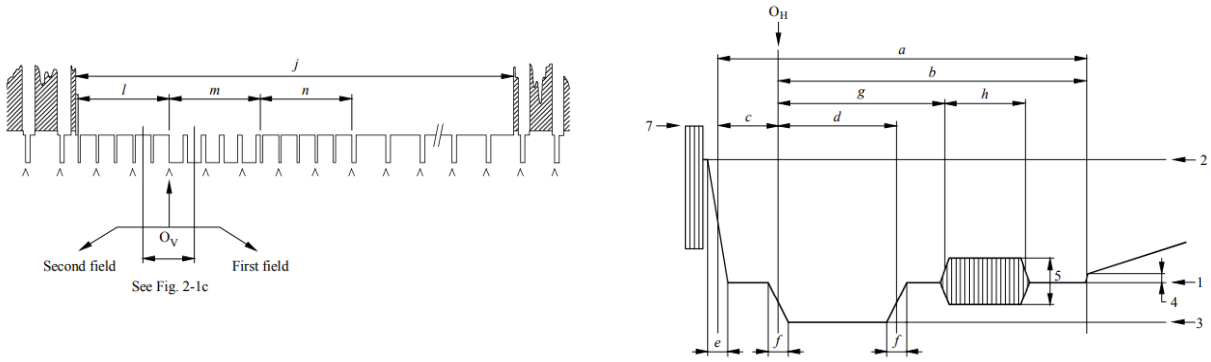


Figure 7: PAL synchronization signals, see `R-REC-BT.470-6-199811.pdf` p.4-p.6

And here are the results obtained :

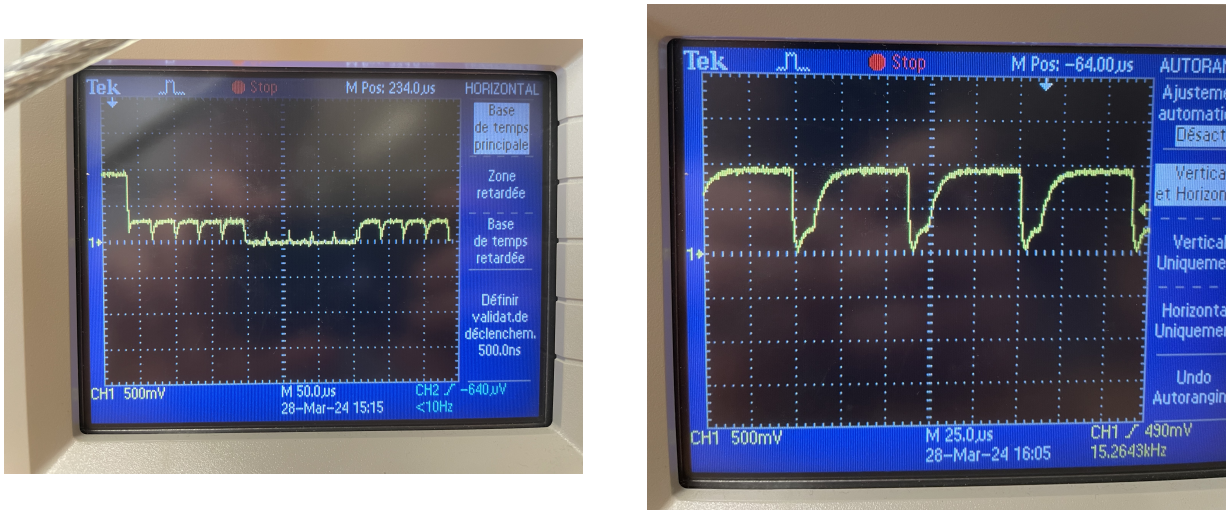


Figure 8: On the left the field synchronization, on the right the scan-lines of a field

As discussed in the `Challenges` section, the signal strength of the signal generated by the `DAC1` module of the `PIC16f1789` was not enough for the video application. The use of an operational amplifier was required to display the white screen in the next test.

3.4.3 Generation of a blank image on a screen

To generate a white image on the screen, we used the operational amplifier module `OPA1` of the `PIC16f1789`.

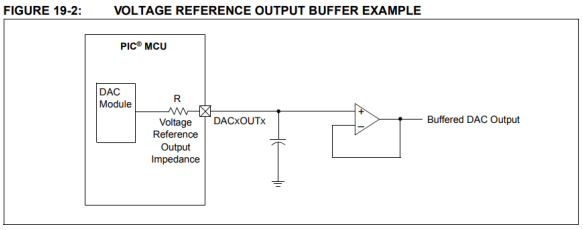


Figure 9: On the left the electrical assembly, on the right the white screen.

To implement the circuit above, OPA1IN+ is connected in software to DAC1OUT1 (RA2), OPA1IN- (RA5) is connected in hardware to OPA1OUT (RA1) and the video connector is connected in hardware between ground and RA1.

4 Challenges

4.1 PAL signal generation using only the DAC1 module

4.1.1 Description

In the first version, the software relied only on the DAC1 module for the video signal generation. The result was that nothing was showing up on the TV screen.

4.1.2 Diagnostic

- A first read of the TV documentation confirmed that the configuration and the connector used were correct.
- An analysis of the signal on the oscilloscope confirmed that the signal generated was correct in terms of timings and voltage levels.
- A reading of the DAC1 documentation, more specifically the `figure 20-2` of the datasheet highlighted the need for some sort of signal amplification.

4.1.3 Solution

After more discussion with the assistant and the professor, it was discovered that the impedance of the TV of 75 Ohm was affecting the signal.
Indeed, even tough the signal on the oscilloscope was clear when connected directly to the DAC1 output pin, it was not the case anymore when connected to a 75 Ohm resistor in series with the DAC1 output pin.

Thus, a follower circuit like the one showed in the datasheet `figure 20-2` was mandatory to amplify the signal. Very practically, it just so happens that the `PIC16f1789` has one in the module OPA1. After adding the OPA1 between the DAC1 output and the video connector, the system behaved as expected and the white screen showed up as seen in Generation of a blank image on a screen!

4.2 PAL signal generation using only the DAC1 and OPA1 modules

In this second version, the software relied on the DAC1 and OPA1 modules for the video signal generation. The system worked fine for a simple white screen but we could already see a first problem in Generation of a blank image on a screen : a black rectangle was appearing on the right of the screen.

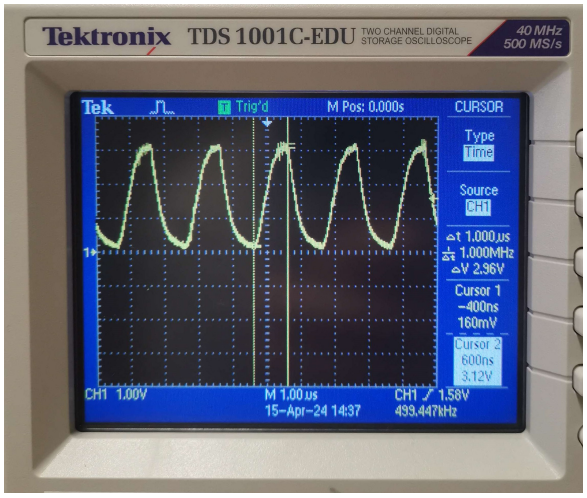
4.2.1 Diagnostic

The first supposed source of the problem was that the PAL sequences were not correctly generated (sync missing, sync of wrong size, wrong voltage value,...), resulting somehow in a shift of the image to the left of the screen.
It was decided to separate the screen in four vertical section of different colors to have a better vision of how the screen was divided. The result is visible on the right picture.

It is clear that the first rectangle on the left begins way before the left corner of the screen. The two rectangles on the center have the same size but were larger than expected and the rectangle on the right was also larger than expected. Another observation (harder to see in the image) was that the transitions between the colors were slow. The visible transitions between the colors measured 1cm, i.e. 0.75µs. Indeed, the screen is 68cm wide, so 1cm = 52 * (1/68) = 0,75µs long.

4.2.2 Further diagnostic

If the slow visible transitions were the result of the limitations of the DAC1/OPA1 modules, the video signal generation would have to be upgraded to something faster. To verify that DAC1/OPA1 were to blame, a HIGH/LOW signal was generated. The signal was programmed to stay at 0V during 1µs, then climb to 3V during 1µs, then 0V, etc. The result can be seen on the right.



The signal is expected to be squared but it is not the case in the oscillogram above. The desired voltage (0V or 1V de) is reached only during around 0,1µs. Which confirms that the modules DAC1 and OPA1 are too slow for the video application.
After discussion with the professor, it has been found that the slew rate of the OPA1 module is to blame. As it can be seen in `table 31-15` p.413 of the PIC datasheet, the slew rate of the OPA1 module is only 3V/µs. This data is in accordance with the result above.

4.2.3 Solution

After more discussion with the professor and some research, the signal video generation with the DAC1 and OPA1 modules was replaced by the use of two digital outputs combined with resistors to produced 4 voltage levels, see Video system section for more explanation about the final solution.

4.3 Compiler core dump

During the development phase, an error has been encountered from the compiler :
`"/opt/microchip/xc8/v2.46/pic/bin/aspic: signal 11 - Core dumped"`. At first it was thought that the problem came from the project configuration, thus it has been tried to run the code on : 3 different computers, MPLab v6.2 and V6.15, Windows and Linux. The program ran correctly on Windows but not on Linux.

To investigate further, parts of the program have been removed until it compile again. It was expected that something "rational" had to be removed, i.e. that something has been done incorrectly in the program. Which was not the case. A folder `/core_dumped` containing multiple solutions is joined to the report.

The archive contains several folders:

- `/compilation` contains the Makefile for compiling and the (old) project files (except game_logic.s).
- `/game_logic_crash` contains the initial version of game_logic.s, (to be placed in /compilation to compile) which produces the core dump in aspic.
- `/game_logic_fix_nop` contains an initial correction which consists of adding a nop after the label.
- `/game_logic_fix_move` contains a second correction, which consists in moving the claim_x section above its comment.
- `/game_logic_fix_comment` contains a third version which consists of changing a word at a very specific position in a comment.

After exchanging with the professor, it has been confirmed that it is a bug of the compiler. It appears that removing the `-Wa,-a` option (used to generate the listing files) solves the issue.

5 Key results

The final program is a complete game with two songs, a custom map, a "hearts" system, a score and best score. The result reaches the expectations, the game is fairly fun to play and user-friendly.

Figure 10: On the left the menu on start, on the right the menu after a new "high score".

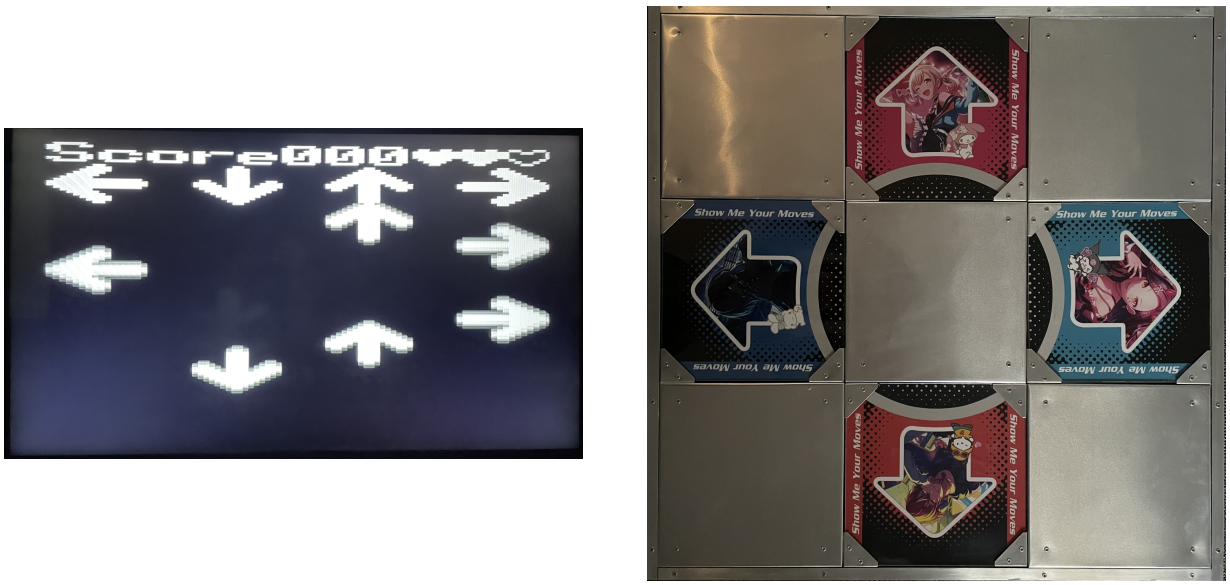


Figure 11: On the left the game in action, on the right the pad.

6 Improvements

6.1 Visual

Our DAC is able to produce grey in addition to black and white. It could be possible to add more shades to our visual output if we had more RAM to store the frame buffer. Currently we use 1728 bytes to store the whole frame with a resolution of 96 pixels $\times$ 144 pixels. Using 1 bit per pixel to store the color is possible if we have only two colors. To add grey in our program we would need at least more than half of the current memory needed to store this buffer and two times the current memory to make things easier with two bits per pixel instead of one and a half.

6.2 Gameplay

Our game only features one music with the same arrow pattern every game. It should be possible to make different musics with their own arrow patterns with more time.

Furthermore, an arrow pattern where the player would have to keep pressing the pad for some time could be added.

We could also implement difficulty mode and based on the level the arrows/music can go faster or slower.

7 Datasheets and documentation

The datasheets of all the components / documentation of the standards (i.e. PAL) used in this project are attached to the report, in `/documentation`. The documents are named by the reference of the component.