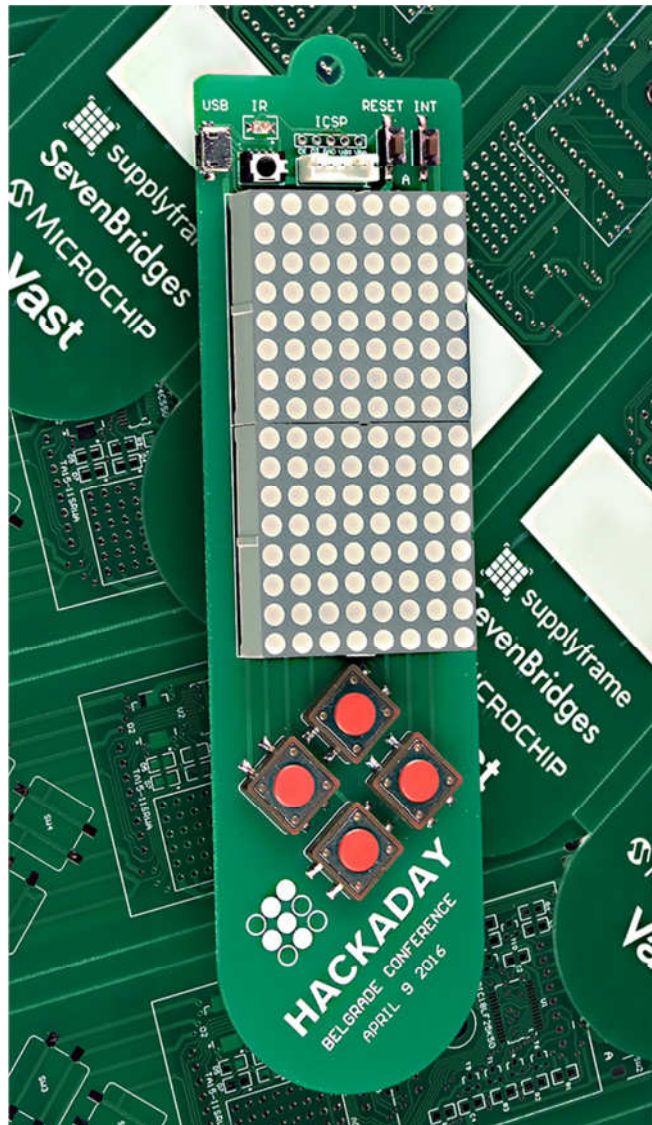


# HACKADAY BELGRADE CONFERENCE BADGE



# 1. INTRODUCTION

Hackaday Belgrade Badge is a special gift for all Hackaday Belgrade visitors. We decided to make it into a bit of a hacking tool, so that everyone gets a chance to display their creativity and programming skills during the conference.

Hacking will be mainly through (but not limited to) software and, but since the badge is actually an embedded system, it does need a special programming process. To program the embedded system, you normally need programmer hardware, but not in this case - you don't need any hardware to flash your own code in MCU program memory. The badge comes with a pre-programmed bootloader, so all you need is a USB Micro-B cable and bootloader software for PC.

To keep the badge "alive" during the conference and before the hacking night, it comes with a couple of pre-programmed applications, mainly to demonstrate some of the possibilities of the hardware. By default, there is a Tetris game and a moving message display. There will be a special computer with infrared interface at the venue, which anyone can use to load a custom message to their own badge. We encourage everyone to keep their badge in Display mode with their own message, so that we can all enjoy the scene of hundreds of personal moving messages running around.

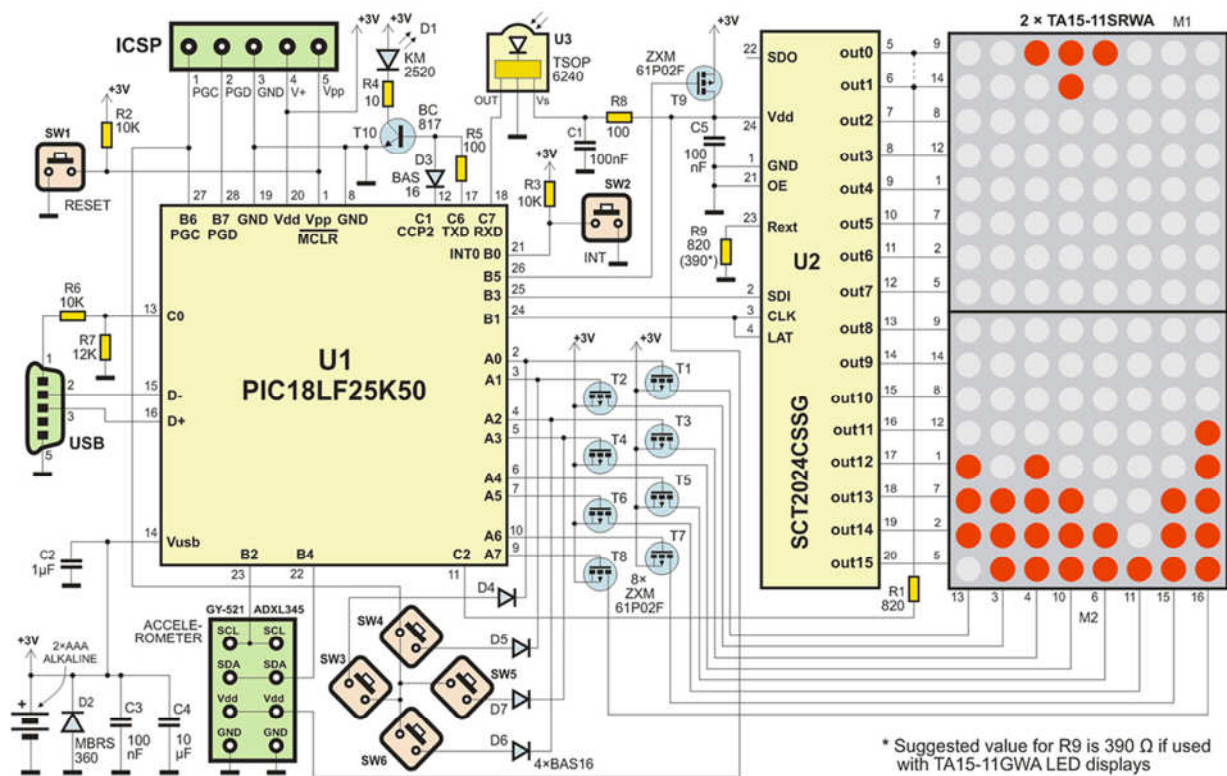
We will also be hosting a puzzle event. Special task badges, with which you can communicate through infrared, will be around the place. All visitors are invited to try to solve the problem during the hacking night, which will consist of trying to make any one of these task badges display a specific message in order to win a prize. All of the task badges will be identical.

## 2. HARDWARE

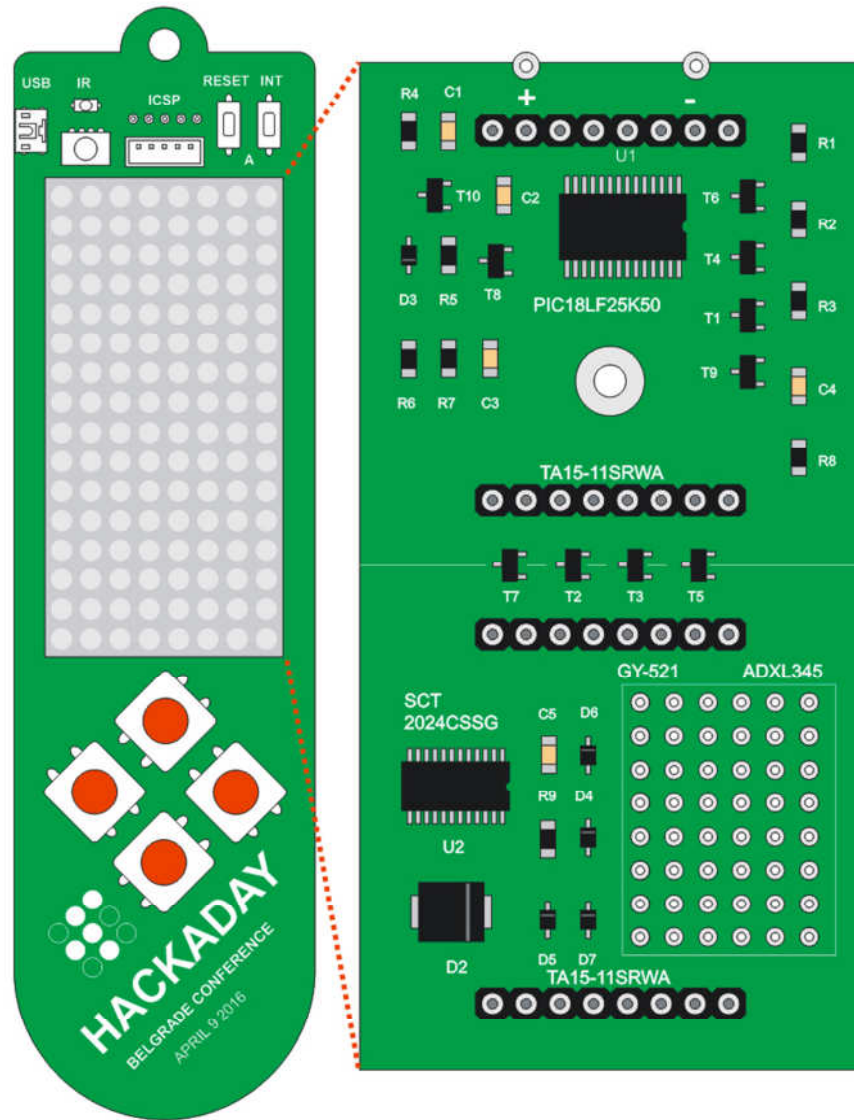
The display of the badge consists a red 8×16 LED matrix. There is also an infrared transceiver and five tactile buttons, plus a RESET key. MCU is Microchip's 8-bit PIC18LF25K50 in 28-pin case, and the power is supplied by two AAA batteries. There are two connectors: USB Micro-B and 5-pin (DW male DIP Header, 2 mm pitch) In-Circuit Serial programming port, convenient for Microchip's PICKIT3 or any other ICD or programmer.

PCB dimensions are 48×176 mm. 8×16 LED matrix is built of two TA15-11SRWA blocks and refreshed by SCT2024CSSG constant current driver, permanently assisted by MCU. CPU clock is 48 MHz (which is 12 MIPS), so display refresh takes about 1% of processor time. Infrared transmitter is a single 940 nm LED, and the receiver is TSOP6240TTCD, which contains a photo detector, an AGC preamplifier, and a 40KHz band-pass filter and demodulator.

Here is the complete schematic diagram of the badge:

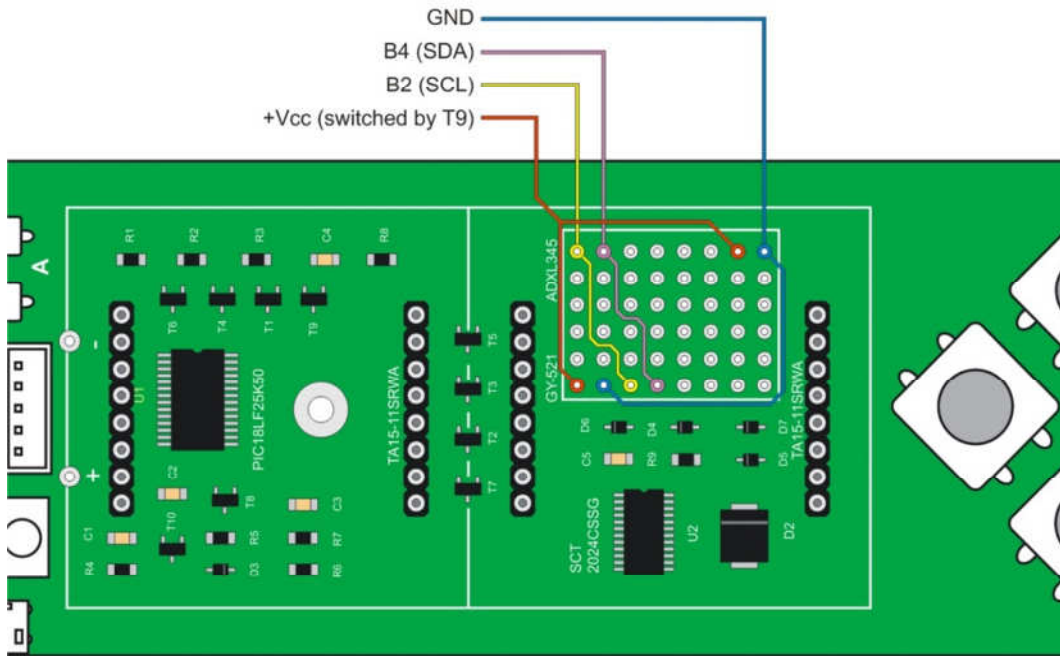


And here follows the PCB layout with LED displays inserted, and a zoomed in view of the area under the displays:



All parts are located on the front side of the PCB, except for the batteries which are on the bottom layer. Connectors, switches and infrared transmitter unit are exposed, but all ICs and passive components are located behind LED display blocks and are not visible to the user. The display blocks are in the sockets, so they are easily removable.

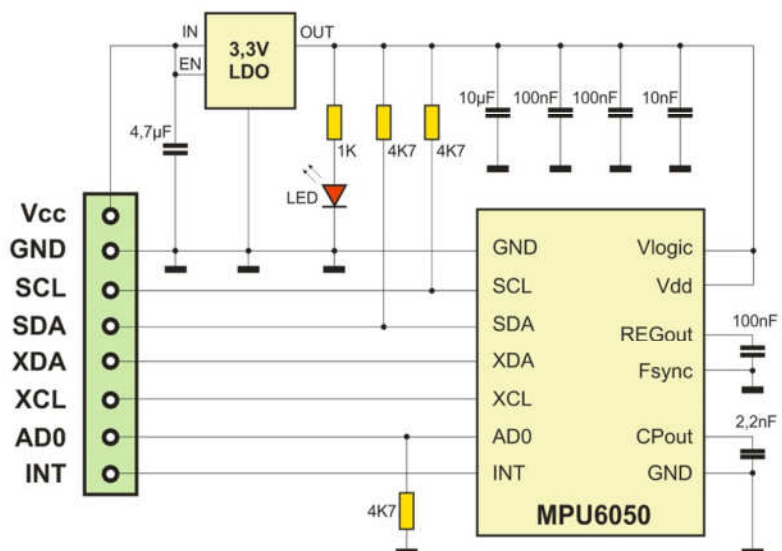
There is a prototyping area with 8x6 soldering pads at the right side under the lower display block. Some of those pads are connected to Vcc (switched by T9), GND and port pins B2 and B4. Those pins are arranged so that accelerometer/gyroscope module GY-521 or ADXL345 may be easily connected.



NOTE:

The accelerometer module is not provided with the badge. If you want to use it, please bring one with you. Suggested type is GY-521.

Here is its internal schematic of GY-521 module. Only the upper four pins should be connected.



## 2. SOFTWARE

The badge comes pre-programmed with three basic firmware modules:

- Bootloader, which normally stays resident and ready for loading any other .HEX firmware via USB cable only
- Kernel, which may (if wished so) be used for new firmware developing
- Demo application. Kernel and Demo firmware modules are linked and loaded by the Bootloader, so if Bootloader is used to burn in new firmware, they will automatically be cleared from memory

Of course, you can also use your PIC programmer if you wish to start programming the MCU from scratch. That way you won't need the bootloader, but if you want to use the Kernel source file to support your application, you'll have to rearrange it at some critical points, predominantly its ISR vectors. The easiest way to do it is to change its Offset variable, which is defined in **Macros.inc** file, and to comment out the line

```
#include <BootLoader.inc>
```

in the **Kernel.asm** file.

The whole project (hardware and all source files for all modules) is open, so you are free to use it, copy and modify and redistribute it.

### 2.1. CONFIGURATION BITS

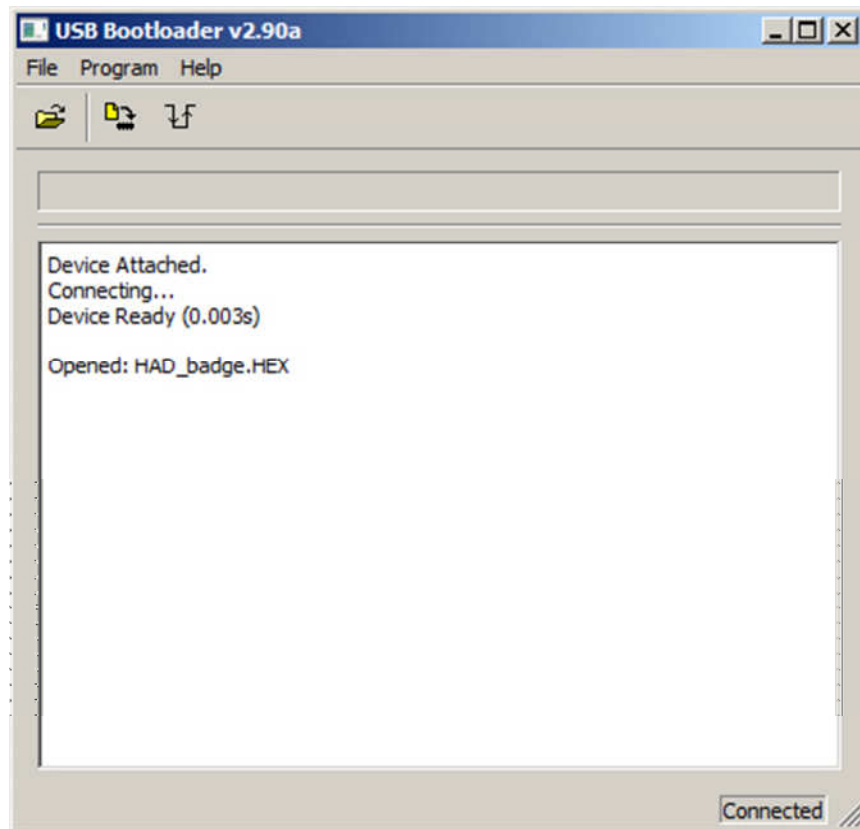
- Internal RC oscillator 16 MHz (primary oscillator disabled)
- PLL with 3× clock multiplier (3×16=48MHz)
- System clock at 48 MHz (which is 12 MIPS operation)
- Brown Out Reset disabled
- Watchdog Timer disabled
- MCLR pin enabled
- Extended instruction set disabled
- Single-Supply ICSP disabled
- Background debugger disabled, B6 and B7 are I/O pins

It is possible to change configuration bits via Table Write process, but note that it may be critical, as some other oscillator settings may render the USB interface and bootloader unusable. In that case, the only way to unlock the unit is to reprogram it with an external programmer.

## 2.2. BOOTLOADER

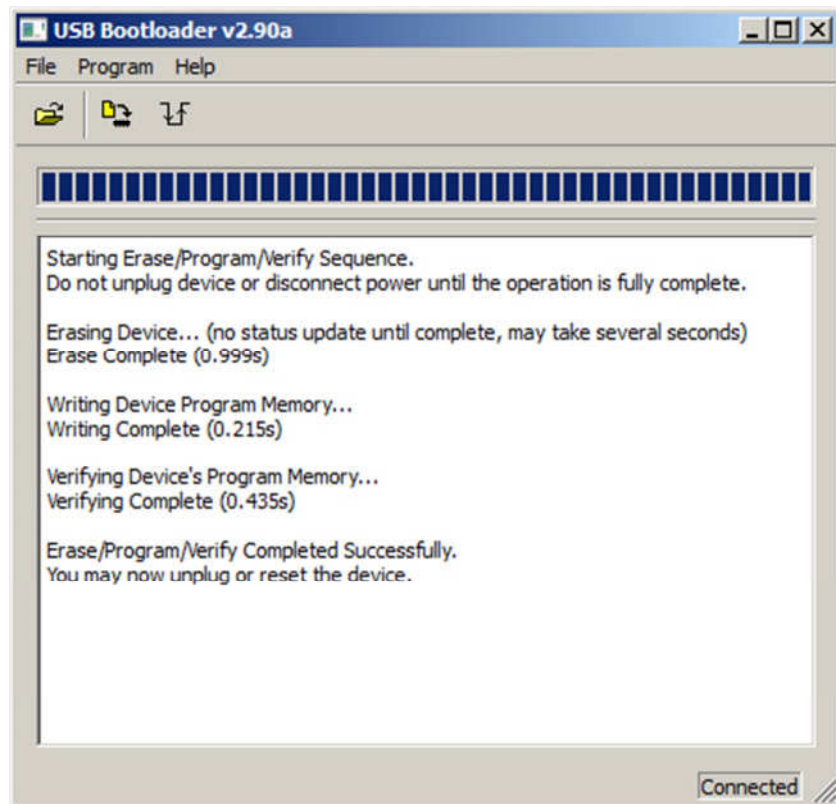
There is a Microchip's MCHPFSUSB bootloader at MCU side, and USB HID Bootloader v2.90a for Windows at PC side. Here is an example of the process of program memory flashing:

1. Write your program and compile (or assemble) it to make a HEX file
2. Connect the badge to your PC using USB Micro-B cable
3. Check that the badge is properly connected to the USB port and that it does not execute user program (press RESET button). The unit should be in Bootloader mode, and one LED should blink.
4. Open MCHPFSUSB Bootloader and load your HEX file



5. Click the Program icon. The taskbar should display progress, but if that does not happen, you have to wait for its internal timeout, which will last for about  $2 \times 10$  sec. Then check all connections, press the RESET button on the badge, and try again.

This is the desired Bootloader output:



MCU accepts the HEX file and self-programs itself. As the bootloader must stay intact in program memory range 0x0000-0x0FFF, your program should have the origin at 0x1000. Bootloader will redirect all vectors (Reset, Low priority interrupt and High priority interrupt) to new addresses, with 0x1000 offset. Bootloader will stay resident in MCU program memory, so it is ready for new program flashing. Every new programming clears the previously programmed version.

After the programming, you can simply press INT button on the badge to start program execution. Alternative way to do it is to disconnect the USB connector and to press the RESET button on the badge. Every time the badge is reset, the bootloader tests if USB voltage is present. If it isn't (if input port C0 is at low level), it jumps to the standard offset address 0x1000.



## 2.3. MEMORY ORGANIZATION

PIC18LF25K50 contains 32 K Program Flash memory, 2 K Data RAM memory and 256 bytes Data EEPROM memory. Program Flash memory and Data RAM memory are directly accessible, but Data EEPROM memory acts as a peripheral device, since it is accessed through a set of control registers.

Kernel uses Data EEPROM memory as non-volatile storage for Brightness control (1 byte) and as a buffer for data received via the infrared communication channel (255 bytes).

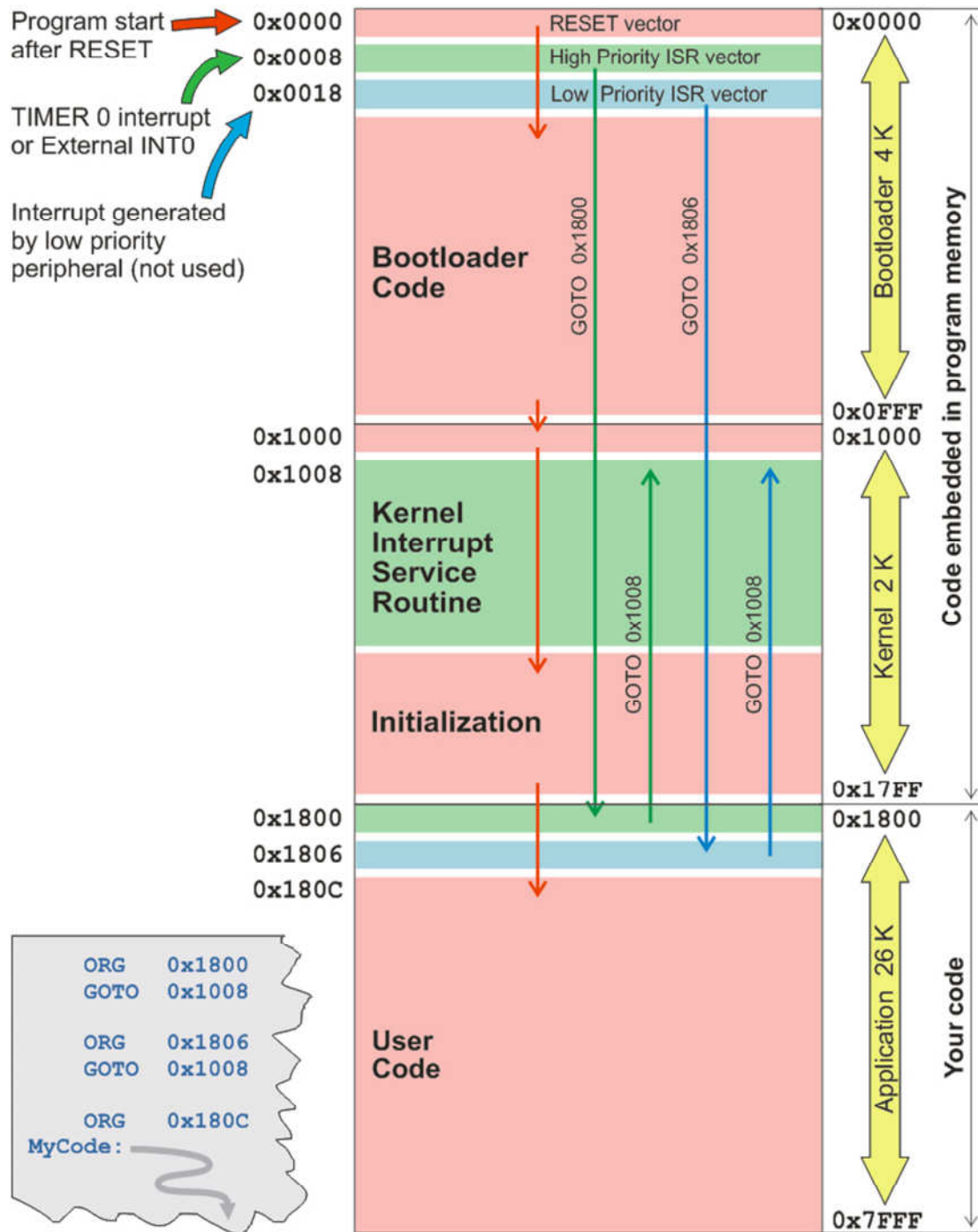
### 2.3.1. PROGRAM MEMORY ORGANIZATION

Bootloader and kernel are integrated and they are used as the main firmware module, taking 6K of MCU's program space. User application must start at 0x1800 and must contain redirection vectors for High Priority ISR (Interrupt Service Routine) at the address 0x1808, and for Low Priority ISR at the address 0x1818. By default, they should contain jumps to 0x1008, where the ISR routine which does most of the Kernel job is located. Low Priority ISR is not used in this version of Kernel, so it also points to the High Priority ISR, but this jump should never happen.

Another way is to redirect those interrupt vectors to the alternative user ISR routine, located inside the User Code area. In that case, the new ISR code should be written, or it may act as the expansion of the existing High Priority Interrupt.

There is only one subroutine available to the user. It is a 32-bit pseudorandom number subroutine, and its entry point is at the address 0x1004. The execution time for this subroutine is 79 T cycles (which is 6.6  $\mu$ s), including Call to this subroutine and Return. It uses Data RAM locations 0x730...0x733 as 32-bit SEED, and locations 0x734...0x737 as the internal arithmetic registers. At the end, before RETURN, this subroutine XORs and ADDs all SEED bytes to produce the pseudorandom number in W register. Also, it XORs contents of TMR0 and TMR2 Special Function Registers in a single 8-bit random result in W register.

The following drawing represents program memory organization of the bootloader, kernel and user application.



### 2.3.2. DATA RAM ORGANIZATION

Kernel uses Data RAM GPR (General Purpose Register) area locations 0x000...0x01B for housekeeping registers. Locations 0x600...0737 are used for special purpose.

<i>addr</i>	<i>name</i>	<i>bit</i>	<i>description</i>
0x00	KeyEdge	0 = set by kernel if key INT edge detected (user must reset) 1 = set by kernel if key LEFT edge detected (user must reset) 2 = set by kernel if key UP edge detected (user must reset) 3 = set by kernel if key DOWN edge detected (user must reset) 4 = set by kernel if key RIGHT edge detected (user must reset) 5...7 Not used	
0x01	Rotor0	Used for key INT debouncer (rotate left, bit = 0 if key pressed)	
0x02	Rotor1	Used for key LEFT debouncer (rotate left, bit = 0 if key pressed)	
0x03	Rotor2	Used for key UP debouncer (rotate left, bit = 0 if key pressed)	
0x04	Rotor3	Used for key DOWN debouncer (rotate left, bit = 0 if key pressed)	
0x05	Rotor4	Used for key RIGHT debouncer (rotate left, bit = 0 if key pressed)	
0x06	Flag	0 = set by kernel: Pause, clr by kernel: Run mode (read only) 1 = Timer 0 interrupt (1200 Hz) handshaking (user must reset) 2 = Full display scan (150 Hz) handshaking (user must reset) 3 = set: EEPROM RX buffer function disabled 4 = Timer 0 interrupt in 2nd phase (LED OFF period) (read only) 5 = set: Disable pause mode (set/clr by user) 6 = Flag that display message was received 7 = Column cathode display (read only)	
0x07	RXFlag	0 = set: Enable auto RX to RAM 0x600...0x6FF and to EEprom 1 = set: RX header reception is in progress (read only) 2 = set: RX message reception is in progress (read only) 3 = set: RX message received (internal use, do not modify it)	

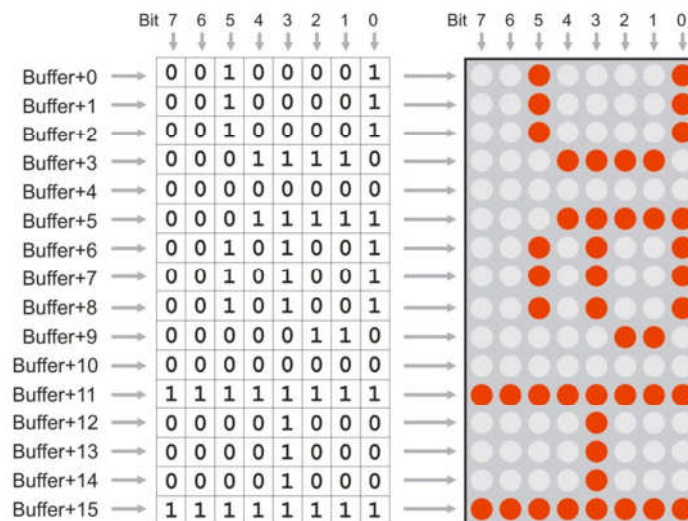
4...7 Not used

0x08	Brightness	Display ON PWM, user presets to 0...15 for dimming
0x09	GPreg	General purpose register, may be used by user
0x0A	BitMask	10000000...00000001, shift reg (CA) or 0...7 row count (CC)
0x0B	BitMask2	11111110...01111111, shift reg used for anode scan
0x0C	T0period	Total Timer 0 period, may be modified to alter scan frequency
0x0D	InnerInt	Loop counter, used by interrupt routine
0x0E	OuterInt	Loop counter, used by interrupt routine
0x0F	OuterPlusInt	Loop counter, used by interrupt routine
0x10	RXptr	Low RXD buffer pointer (high is always =6)
0x11	RXpatience	Patience counter, preset when byte received, count down
0x12	PowerOFF	Auto Power Off period (×6 sec), preset here to alter timing
0x13	PowerCount	Auto Power Off count down
0x14	Uniform	(2 bytes) 150 Hz divider, count up for 6 sec timing
0x16	RXserial	(2 bytes) Received serial number (binary), ready for comparison
0x18	MySerial	(2 bytes) unit serial number copied from ROM address 0x100E
0x1A	FSR0temp	(2 bytes) Temporary FSR0 during INT
0x1C...0x5FF		User data RAM space
0x600...0x6FF		RX Buffer, used by infrared port routine (bytes loaded here)
0x700...0x70F		Display buffer, upper row first, bit 7 = left column (user writes here)
0x710...0x71F		Aux buffer (not displayed by interrupt display refresh, used by user)
0x720...0x72F		Pause display buffer (displayed only during pause)
0x730...0x733		RND seed
0x734...0x737		RND internal arithmetic registers

### 3. KERNEL

Kernel supports LED matrix multiplex. It also contains an initialization routine (which is normally executed only once after RESET) and Timer 2 Interrupt routine, which should always be active. This routine executes uniformly at 1200Hz rate in 8 steps, so it enables a 150Hz display refresh rate. Within this routine, there is a key scanning subroutine with a debouncer and an edge detector, and full ON-OFF-Pause control. So, MCU sleeps in Interrupt routine and user does not have to take care of that. There is also UART RX manager, which automatically loads received string in RX buffer (0x600-0x60E), if all conditions are met.

Frame buffer is in RAM, and everything that the user writes in 0x700-0x70F will be immediately displayed on the LED screen. There is one more auxiliary buffer, which is not displayed and is free to be used by the user routine. The third buffer (0x720-0x72F) is a special frame buffer which will be displayed only in Pause mode. It may be useful for score displaying, pause symbol or any message.



#### 3.1. KERNEL OUTSIDE INTERRUPT

The part of kernel which is outside of the interrupt routine, is located in the file kernel.asm. It also includes files p18lf25k50.inc (with processor definitions), macros.inc (with macro definitions) and int.inc (the part of kernel which is executed in interrupt).

First, it presets Special Function Registers:

- OSCCON and OSCCON2: Sleep mode enabled, all other bits are unchanged, as defined in configuration words

- ANSEL: all inputs all digital, except C2, which is AN14 (Note: ANSEL registers are in BANKED address space!)
- INTCON2: Internal pull-ups enabled, int0 on falling edge of PORTB,0 input. This interrupt will be used inside TIMER 0 interrupt routine, for wake-up after sleep
- WPUB: Only PortB,6 pull-up enabled (that's key1...key4 input, driven by A0...A3 output ports)
- LATx and TRISx bits preset as hardware requires
- T0CON: Timer 0 defined as 8-bit timer, prescaler = 128, software interrupt on overflow. This interrupt is used for LED display refresh support, with dynamically adjusted timing for display dimming (LED intensity control)
- T2CON: Timer 2 with 1:4 prescaler and no postscaler. Used for PWM peripheral which generates a 40 KHz carrier for infrared transmitter. PWM peripheral is defined with CCP2CON (which selects PWM mode) and CCP2L which defines signal/pulse ratio to approximately 50:50
- TXSTA1, RCSTA1, BAUDCON1, SPBRGH1, SPBRG1: UART TX/RX programmed to 2400 baud, no parity, 8 data bits, 1 stop bit

After register presetting, program erases (presets to 0x00) all data memory, using uninitialized Data RAM to preset RND Seed registers. Four RND Seed registers are preset with random contents, each of them by XORing 512 bytes of Data RAM before erasing.

Next, it loads the binary serial number from program memory address 0x100E (assuming that Offset is preset to 0x1000) to data memory 2-byte location MYserial. If that number is 0xFFFF (default), kernel will ignore it and take the serial number from Bootloader, at address 0x000E. Every badge has its unique serial number defined in the Bootloader, but if you want to redefine it, you can just change the kernel definition - if it is not 0xFFFF, it will be accepted.

Next, it loads the Brightness value from internal EEPROM at address 0x00. This EEPROM contents will be modified every time when brightness is modified by the keyboard.

Next, it loads the display text from internal EEPROM at addresses 0x01-0xFF to data RAM text buffer at fixed addresses 0x600-0x6FE. If the first character is 0x00 or 0xFF, it loads the default greeting message from program memory.

Next, it auto detects the display type (column anode or column cathode) and sets Flag,7 if it is column cathode. This bit will be used in the display driver routine. Auto detection uses the fact that extra cathode terminal of the LED display matrix (which is driven by C2 output in the Bootloader routine and used for LED blinking while the multiplex is off) is wired to Cathode 1 in CC (Column Cathode) version, and to Cathode 2 in CA version. So, the autodetect routine sets output 0 and 1 on cathode driver (SCT2024CSSG) to 1 and 0 respectively first and measures the

voltage response on the AN14 analogue input, as C2 pin is not used as output anymore, but as analogue input. Then it measures the same with 0 and 1 respectively, and then compares measured analogue values. C2 will be left as dummy analogue input for the whole operating time - if you define it as output it will disturb normal multiplex operation, and if it is digital input, the voltage on this pin will be outside of allowed range. At last, it enables TIMER 0 interrupt and jumps to label "cont" which should be defined in application software.

There is also an "rnd" subroutine which is not used by kernel, but can be used by user software. It is a 32-bit pseudorandom generator routine, which executes function  $SEED = SEED * 0x41C64E6D + 0x00006073$ . SEED is defined as Ma0, Ma1, Ma2 and Ma3 in data memory. It also uses (and trashes) arithmetic temporary registers Mc0, Mc1, Mc2 and Mc3 in data memory. At the end, this routine XORs or ADDs all SEED registers and TMR0 and TMR2 also, to scramble the W register, which should be considered as 8-bit random output.

### **3.2. KERNEL INSIDE INTERRUPT**

Interrupt nesting is disabled, so both the External Interrupt 0 (INT key) and TIMER 0 use the same Interrupt vector 0x0008, which is redirected to 0x1808 by the bootloader routine.

Routine first tests INTCON,INTOIF bit to determine if the interrupt was caused by INTO (kernel will not allow such external interrupts except inside TIMER 0 interrupt during processor sleeping, but this test is executed in case user uses it for some purpose). By default, this is a dummy interrupt routine, as it does nothing except reset the interrupt flag.

If the interrupt was triggered by TIMER0 overflow, the routine first presets TMR0L counter to desired timing until the next overflow. This timing depends on the Brightness register, which is in range 0x00 (lowest intensity) to 0x0F (highest intensity). As PWM regulation is used, one Anode period (or 1/8 of total display refresh cycle) contains two interrupts: ON period, and OFF period. Flag,4 is toggled at every interrupt, and it determines if ON or OFF cycle is in progress. In the first cycle, one Anode is active, and in the second one, all Anodes are OFF. Both timings are determined by register Brightness: ON period is  $(\text{Brightness}+1) \times 52\mu\text{s}$ , and OFF period is  $(16-\text{Brightness})$ . The whole period is always 833  $\mu\text{s}$ . Individual timings for each Brightness setting is read from lookup table, so that the intensity regulation is approximately logarithmic.

Display refresh is basically different for CA and CC displays. The first one takes only one bit from all frame buffer locations, and the second one takes subsequent bytes. It is common for both drivers that, at the end, it switches ON only one anode output, whether it is column or row.

If Pause flag (which is in Flag,0) is reset, display routine will output Buffer (16 bytes), and if it is set, BufferPause (16 bytes) contents will be output.

After display driving, Interrupt routine tests all keys (except for the Reset key, of course). Debouncer uses registers ROTOR0...ROTOR4, to shift left each key input. If ROTORx state is = 11111110, the falling edge is detected and debounced (after seven "key off" states) and bit 0...4 in register KeyEdge is set. So, if user routine has to test if some key was JUST pressed (signal transition from HIGH to LOW), it should test one of those bits and reset it after it detects that it was set (is is not automatically reset). If user routine has to test if some key is permanently pressed, it has to test bit 0 in one of ROTOR0...ROTOR4 registers. It is NOT recommended to test port pins directly (except for the INT key), as keys 1...4 have one common input.

Special test is made for Left+Right+Up and for Left+Right+Down keys, as those combinations are used for display brightness adjusting. Left and Right key are tested for permanent pressing (bit 0 in Rotor1 and in Rotor4), and Up and Down keys for edge detection (bit 2 and 3 in KeyEdge register), so it is required that user holds down keys Left and Right at the same time, and presses keys Up and Down to adjust brightness. If those conditions are met, the new Brightness contents are written in Eeprom, at address 0x00. After the Reset condition, on power up, kernel will return this value in register Brightness.

Key INT has the special function. When pressed (which is still not detected via Interrupt process, but via simple polling), kernel will set the Pause flag (which is in Flag,0) and display routine will redirect from Buffer (16 bytes) to BufferPause (16 bytes) contents.

If Pause flag is already set and INT key is pressed, kernel will execute Sleep process. It will switch off all anodes and power supply to all external devices, then pull all other outputs low. Then it will reset INTCON,INTOIF flag, to avoid false wake up and INTCON,TMROIE. It will also set INTCON,INTOIE to enable wake-up. Then it will enter sleep mode.

When the INT key is pressed, external interrupt will wake up the processor and kernel will execute all operations in the inverse order and sense. As external interrupt is impossible to debounce by software, special test is made before sleeping and after wake-up to ensure that INT key is OFF (200ms before and 50 ms after sleeping).

If RXFlag,0 is set by user software, then kernel will automatically receive bytes from the infrared port and write them to the RX input buffer, which is in data RAM, at addresses 0x600...0x6FE. If a string longer than 254 bytes is received, only the first 154 bytes will be detected, and the rest will be ignored.

Message header contains ASCII "[", one to five ASCII digits (which represent recipient's serial number) and ASCII"]". So, header can vary from [0] to [65535]. Only if RXFlag,0 is set, valid





## 4. DEMO APPLICATION

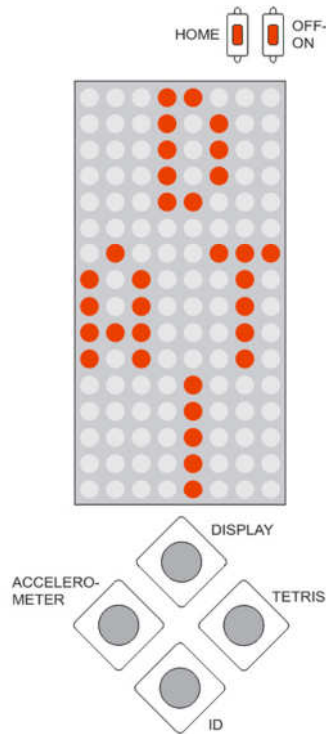
Demo firmware comes loaded by the Bootloader. If you wish to write your own firmware, you will probably have to dismiss the demo firmware module and link your firmware with Kernel. In that case, you will need the following files from the HAD\_BADGE folder:

- kernel.asm
- int.inc
- macros.inc
- p18f25k50.inc
- 18f25k50.lkr

Demo application also employs following files:

- demo.asm
- tetris.inc
- display.inc
- accel.inc

After RESET, the following Home screen is displayed, with the following button functions:



Letters D, A, T, and I are commands for the following programs:

## 4.1. DISPLAY

This is the moving message display program, which runs the ASCII message located at Data RAM addresses 0x600-0x6FF. By default, there is a greeting message, which originally looks like this in the source file:

```
<h>Welcome to<H> Hackaday Belgrade<h> conference <I>H<i> <I>A<i>
<I>C<i> <I>K<i> <I>A<i> <I>D<i> <I>A<i> <I>Y<I> <v>Welcome to<V>
Hackaday Belgrade<v> conference <I>H<i> <I>A<i> <I>C<i>
<I>K<i> <I>A<i> <I>D<i> <I>A<i> <I>Y<I>
```

There are special commands which are not displayed, but used to switch the display mode. Those commands contain one ASCII character enclosed in "<" and ">" delimiters. Here is the list of supported commands (font 5×7 is actually 5×9, but the lower two rows are only used in a few of the lowercase characters):

<h>	Horizontal scrolling, normal font 5×7
<H>	Horizontal scrolling, bold font 8×14
<v>	Vertical scrolling, normal font 5×7
<V>	Vertical scrolling, bold font 8×14
<l>	Lateral scrolling, rotated font 5×7
<i>	Immediate one character with short pause 200 ms
<I>	Immediate one character with long pause 500 ms
<1>...<9>	Pause (number × 250 ms)

The new text can be loaded via the infrared interface. There will be an infrared terminal present during the conference, which anyone can use to type in a custom text message and transmit it to their badge. All listed commands are supported, but the message must have a specific header at the beginning.

This header should consist of ASCII "[", one to five ASCII digits (which represent recipient's serial number) and ASCII "]". That means that the header can vary from [0] to [65535]. When a message is received, the old one is automatically cleared.

Message terminator is pause, which is at least 200 ms long. This means that no special terminator is needed, and the badge will start running the message 200 ms after the whole message is transmitted and there is silence at the infrared communication channel. At the end of the the message, terminator 0x00 will be inserted automatically to the received string. The maximum message length is 254 bytes.

The message is kept in the internal non-volatile EEPROM.

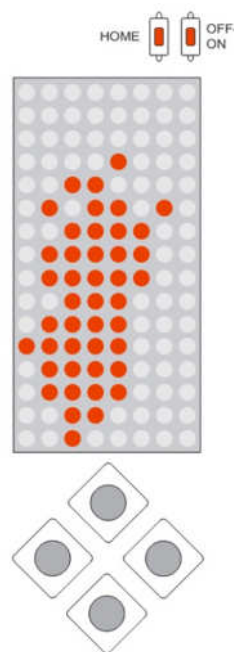


not satisfy all requirements. In some unfavorable conditions, when the signal is weak, it may limit the number of bytes which can be received in one packet, to about 100.

To satisfy all requirements for TSOP6240, user must write his own routine with special format, instead of using the existing UART peripheral.

## 4.2. ACCELEROMETER

Badges are delivered without the accelerometer module, but there is a prototyping area adapted to be able to accept modules GY-521 (recommended) or ADXL345. If you have one of those modules, you can solder them and use this application. The badge will automatically detect which module is present.



The application is simple, it simulates a liquid (or little balls) which are moving as you tilt the badge. You can take a look at the video demonstration at

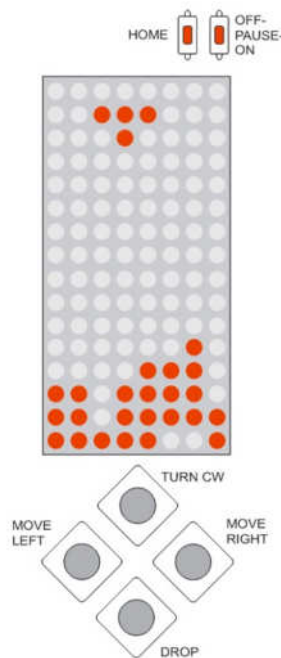
<https://www.youtube.com/watch?v=a2YFsHdKY7c>

## 4.3. TETRIS

This is the old game from the early computer era. "Tetriminos" are game pieces shaped like tetrominoes, geometric shapes composed of four square blocks each. A random sequence of Tetriminos fall down the playing field (a rectangular vertical shaft, called the "well" or "matrix"). The objective of the game is to manipulate these Tetriminos, by moving and rotating them,

with the aim of creating a horizontal line of 8 units without gaps. When such a line is created, it disappears blinking, and any block above the deleted line will fall. When a certain number of lines are cleared, the game enters a new level. As the game progresses, each level causes the Tetriminos to fall faster, and the game ends when the stack of Tetriminos reaches the top of the playing field and no new Tetriminos are able to enter.

Button functions are represented on the following drawing. Game can be paused by INT button, when the score is displayed. INT button also has the OFF function, but game status is not lost in OFF mode. If you press the RESET button, all game parameters are cleared.



#### 4.4. ID

This application doesn't have its own screen. When it is invoked by pressing the lower key, the badge transmits its unique serial number via the infrared transmitter. You can use this function to identify your badge by displaying its serial number on the infrared terminal.

This command is active only in Home mode, which you can tell is active when the letters D, A, T and I are displayed.