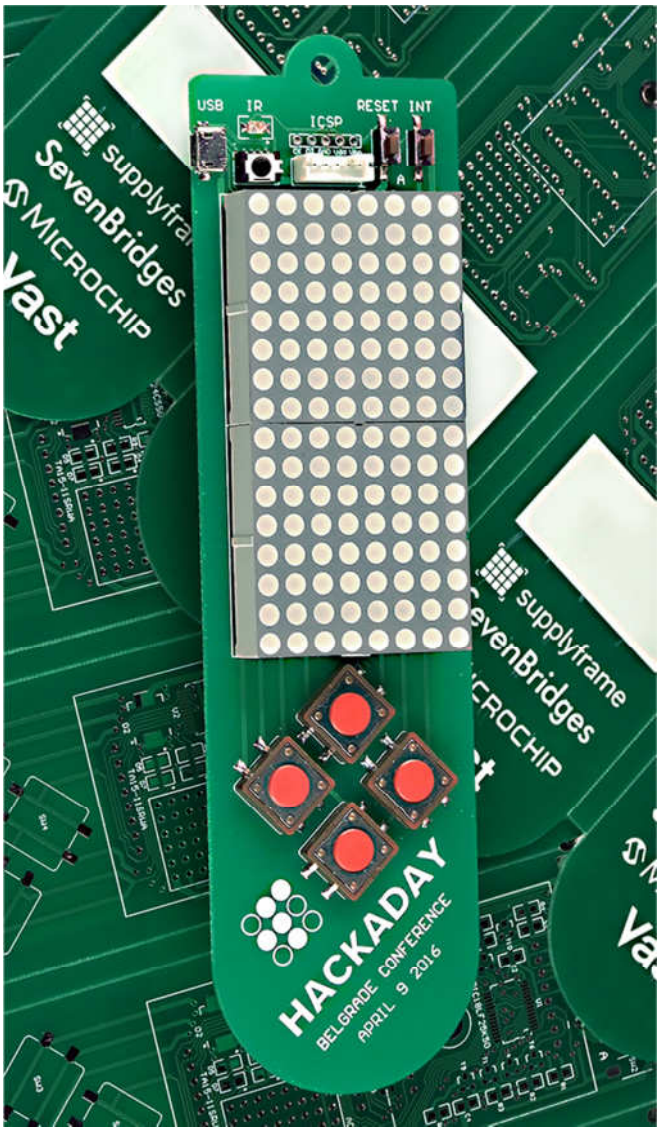
HACKADAY BELGRADE

CONFERENCE BADGE



Ver 1 25. Mart 2016.

1. INTRODUCTION

Hackaday Belgrade Badge is the present to Hackaday Belgrade visitors. During the conference, it shall be used for hacking, so that everybody can express their creativity and programming skill.

Hacking will be mainly in (but not limited to) software and, as the badge is actually the embedded system, it needs the special programming process. To program the embedded system, you normally need the programmer hardware, but that's not the case here: the good news is that you don't need any hardware to flash your own code in MCU program memory. The badge comes with the pre-programmed bootloader, so all you need is USB Micro-B cable and bootloader software for PC, which is available and free.

To keep the badge "alive" during the conference and before the hacking night, there are a few pre-programmed applications which are used demonstrate some possibilities of the hardware. By default, there is the Tetris game and some kind of moving message display. At the conference place, there will be a computer with infrared interface, so everybody will be free to enter his own moving message in his badge. everybody is encouraged to keep their badge in Display mode with their own message, so that we can all enjoy the scene with hundreds of personal moving messages running around.

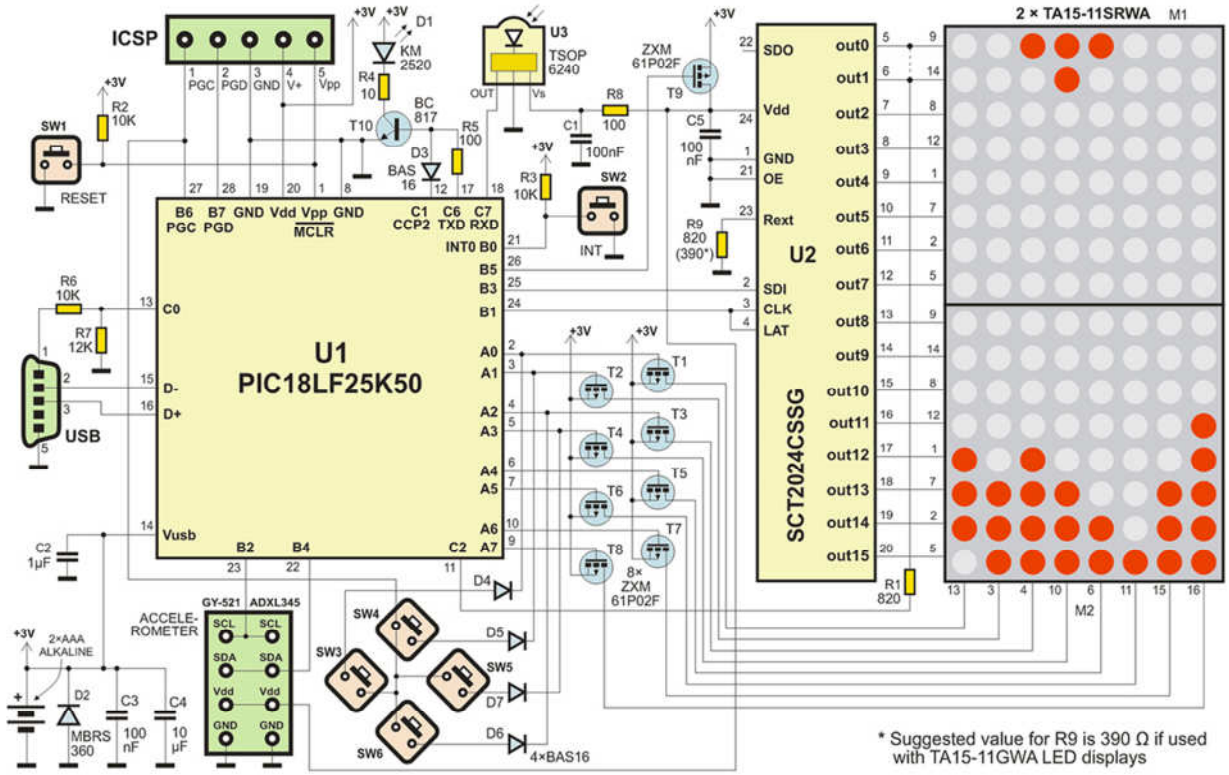
There are also the plans for special task badges which shall be exposed at the conference, so that every visitor can try to solve the problem during the hacking night, and to provoke the badge to display the winning message, and to win a special prize. Infrared communication will be used in this hacking, and it's the only thing that you should know at the beginning - don't ask about the details of the task - it's a part of the task, also!

2. HARDWARE

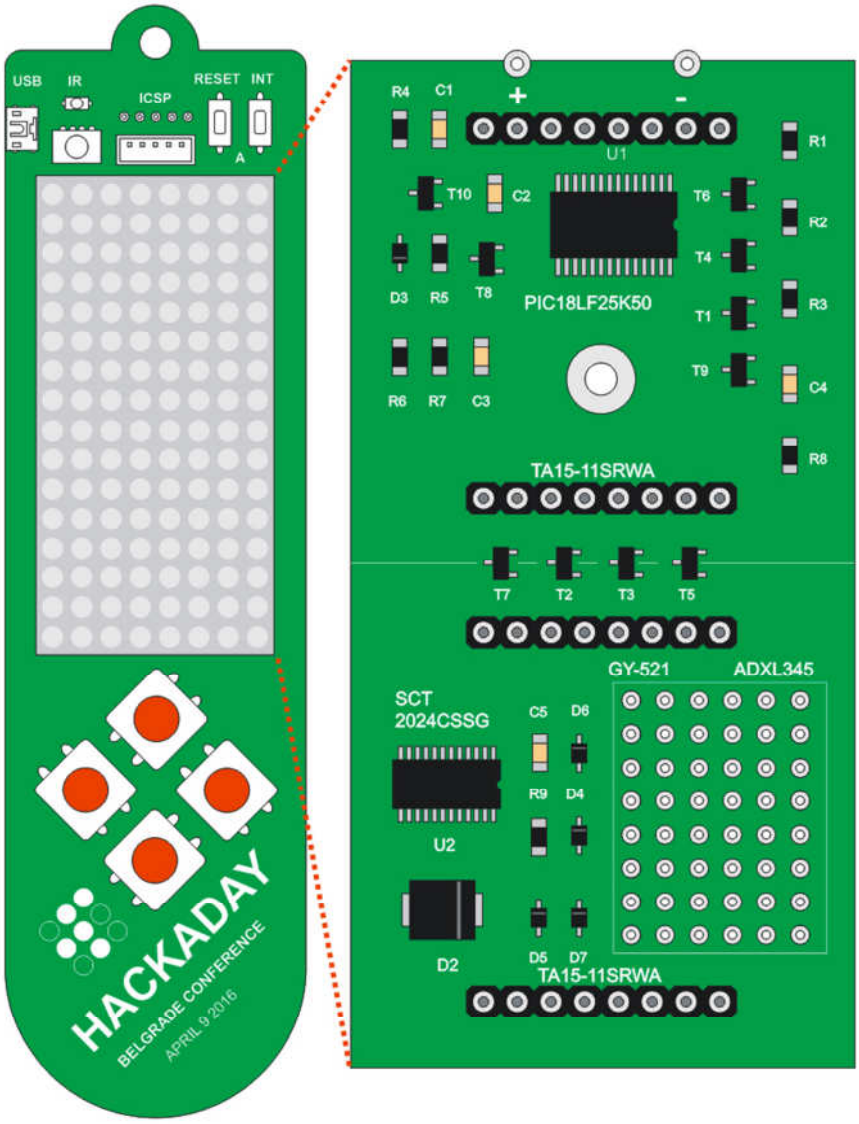
The main part of the badge is the red 8×16 LED matrix. There is also an infrared transceiver and five tactile buttons, plus RESET key. MCU is Microchip's 8-bit PIC18LF25K50 in 28-pin case, and the power is supplied by two AAA batteries. There are two connectors: USB Micro-B and 5-pin (DW male DIP Header, 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 the single 940 nm LED, and the receiver is TSOP6240TTCD, which contains photo detector, AGC preamplifier, 40KHz band-pass filter and demodulator.

Here is the complete schematic diagram of the badge:

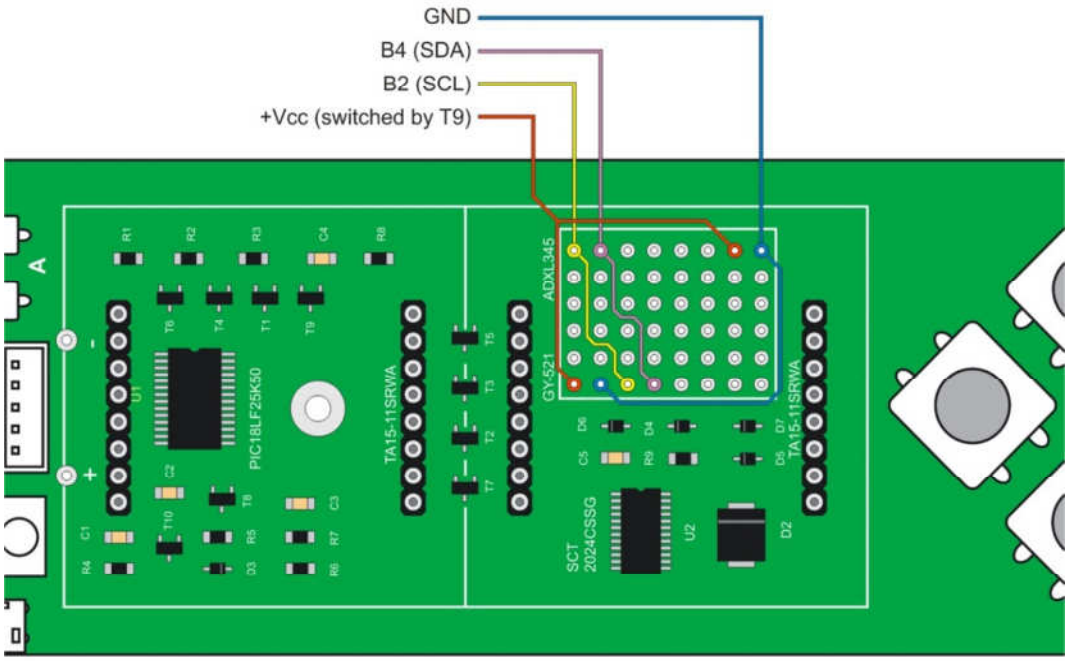


And here follows the PCB layout, with LED displays inserted, and zoomed area under the displays:



All parts are located on the front side of PCB, except 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 not visible to the user. Those blocks are in the sockets, so they are easily removable.

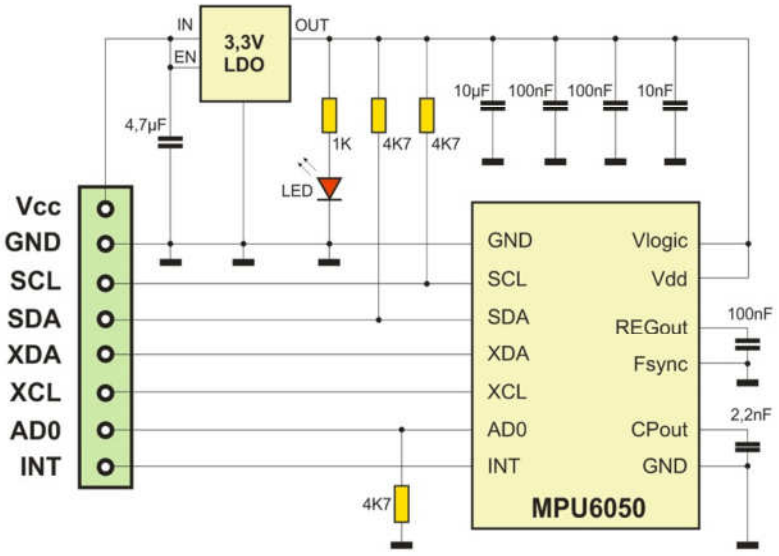
At the right side under the lower display block, there is the prototyping area with 8×6 soldering pads. Some of those pads are connected to Vcc (switched by T9), GND and port pins B2 and B4. Those pins are arranged so that accelerometer/gyroscope module GY-521 or ADXL345 may be easily connected.



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

Badge is initially 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, which is good for pre-hacking demonstration. Kernel and Demo firmware modules are linked and loaded by the Bootloader, so if Bootloader is used to burn in the new firmware, the will automatically be cleared from the memory

Of course, you can also use your PIC programmer if you wish. In that case, you don't need the bootloader.

Source files for all listed modules are available and free.

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 USB interface and bootloader unusable. In that case, the only way to unlock the unit is to reprogram it with external programmer.

3. BOOTLOADER

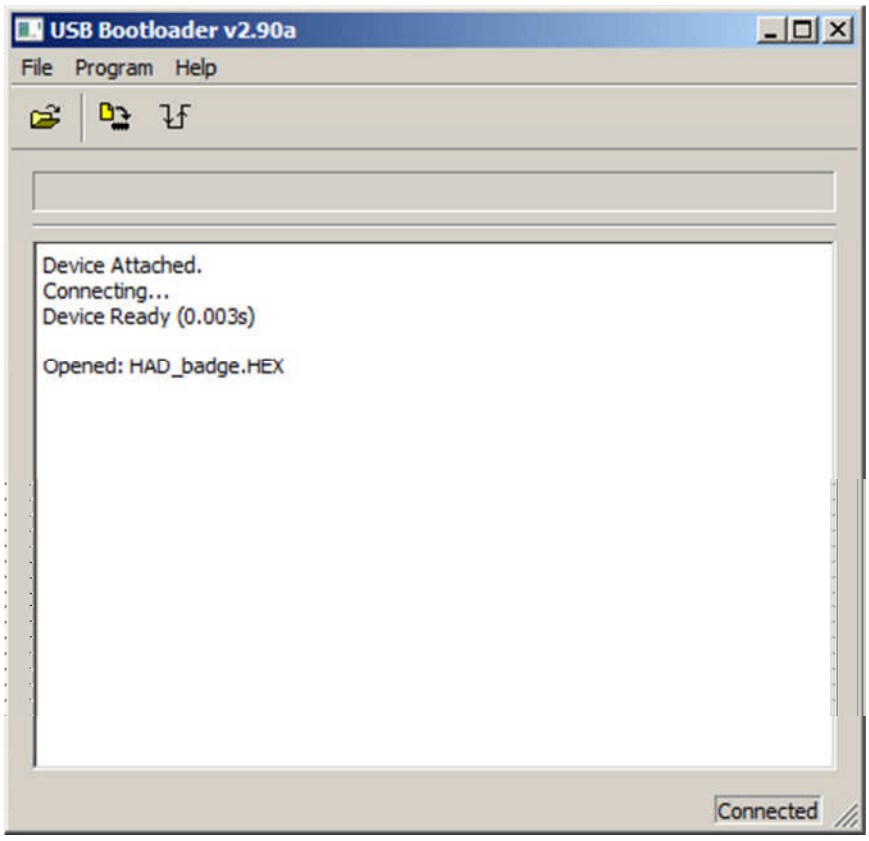
There is Microchip's MCHPFSUSB bootloader at MCU side, and USB HID Bootloader v2.90a for Windows at PC side. Here is the process of program memory flashing:

1. Write your program and compile (or assemble) it to make HEX file

2. Connect the badge to your PC using USB Micro-B cable

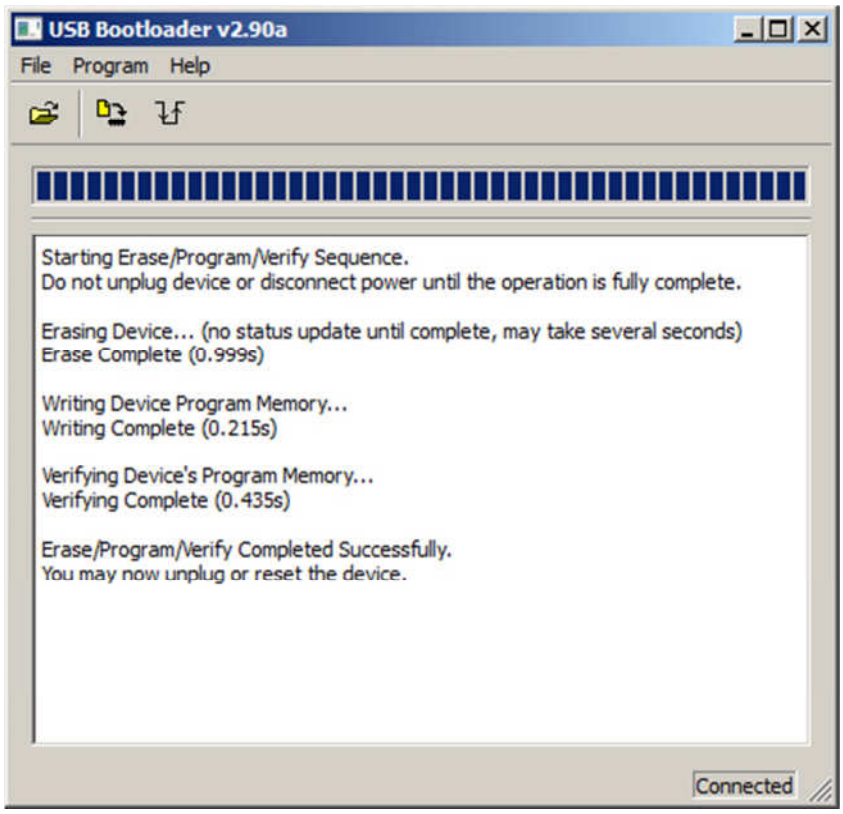
3. Check that the badge is properly connected to the USB port and that it does not execute user program (press RESET button). The unit should be in Bootloader mode, and one LED should blink.

4. Open MCHPFSUSB Bootloader and load your HEX file



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

This is the desired Bootloader output:



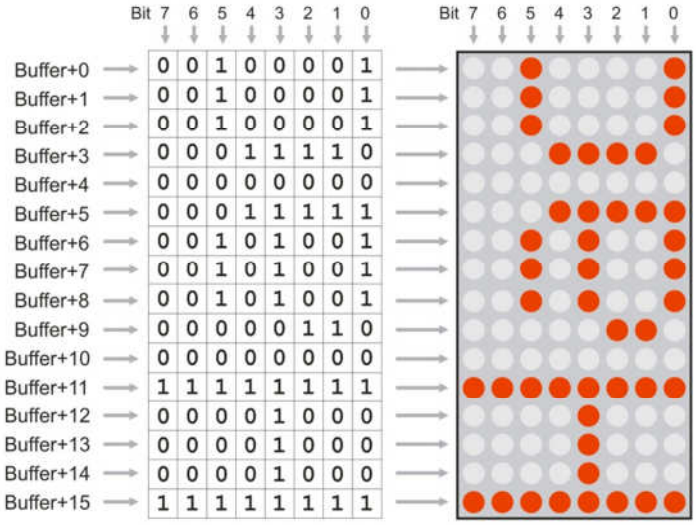
MCU accepts HEX file and self-programs itself. As the bootloader must stay intact in program memory range 0x0000-0x0FFF, your program should have the origin at 0x1000. Bootloader will redirect all vectors (Reset, Low priority interrupt and High priority interrupt) to new addresses, with 0x1000 offset. 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 USB connector and to press 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.

4. KERNEL

Kernel supports LED matrix multiplex, and it contains 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 150Hz display refresh rate. Inside this routine, there is the key scanning subroutine with debouncer and edge detector, and full ON-OFF-Pause control, using the single key. 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 strung in RX buffer (0x600-0x60E), if all conditions are met.

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



4.1. KERNEL OUTSIDE INTERRUPT

The part of kernel which is outside the interrupt routine, is located in 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 40 KHz carrier for infrared transmitter. PWM peripheral is defined with CCP2CON (which selects PWM mode) and CCP2L which defines signal/pause ratio to approximately 50:50
- TXSTA1, RCSTA1, BAUDCON1, SPBRGH1, SPBRG1: UART TX/RX programmed to 600 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 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 display driver routine. Auto detection uses the fact that extra cathode terminal of the LED display matrix (which driven by C2 output in 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

respond on AN14 analogue input, as C2 pin is not used as output anymore, but as analogue input. Than 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 in unallowed range.

At last, it enables TIMER 0 interrupt and jumps to label "cont" which should be defined in application software.

There is also "rnd" subroutine which is not used by kernel, but can be used by user software. It is 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 W register, which should be considered as 8-bit random output.

4.2. KERNEL INSIDE INTERRUPT

Interrupt nesting is disabled, so both External Interrupt 0 (INT key) and TIMER 0 use the same Interrupt vector 0x0008, which is redirected to 0x1008 by the bootloader routine.

Routine first tests INTCON,INT0IF bit to determine if the interrupt was caused by INT0 (kernel will not allow such external interrupt except inside TIMER 0 interrupt during processor sleeping, but this test is executed in case if user uses it for some purpose). By default, this is dummy interrupt routine, as it does nothing except it resets 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 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 (Brightness+1) × 52µs, and OFF period is (16-Brightness). The whole period is always 833 µs. 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 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 edge), 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 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 adjusts 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 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,INT0IF flag, to avoid false wake up and INTCON,TMR0IE. It will also set INTCON,INT0IE 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 infrared port and write them to RX input buffer, which is in data RAM, at addresses 0x600...0x6FE. If 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

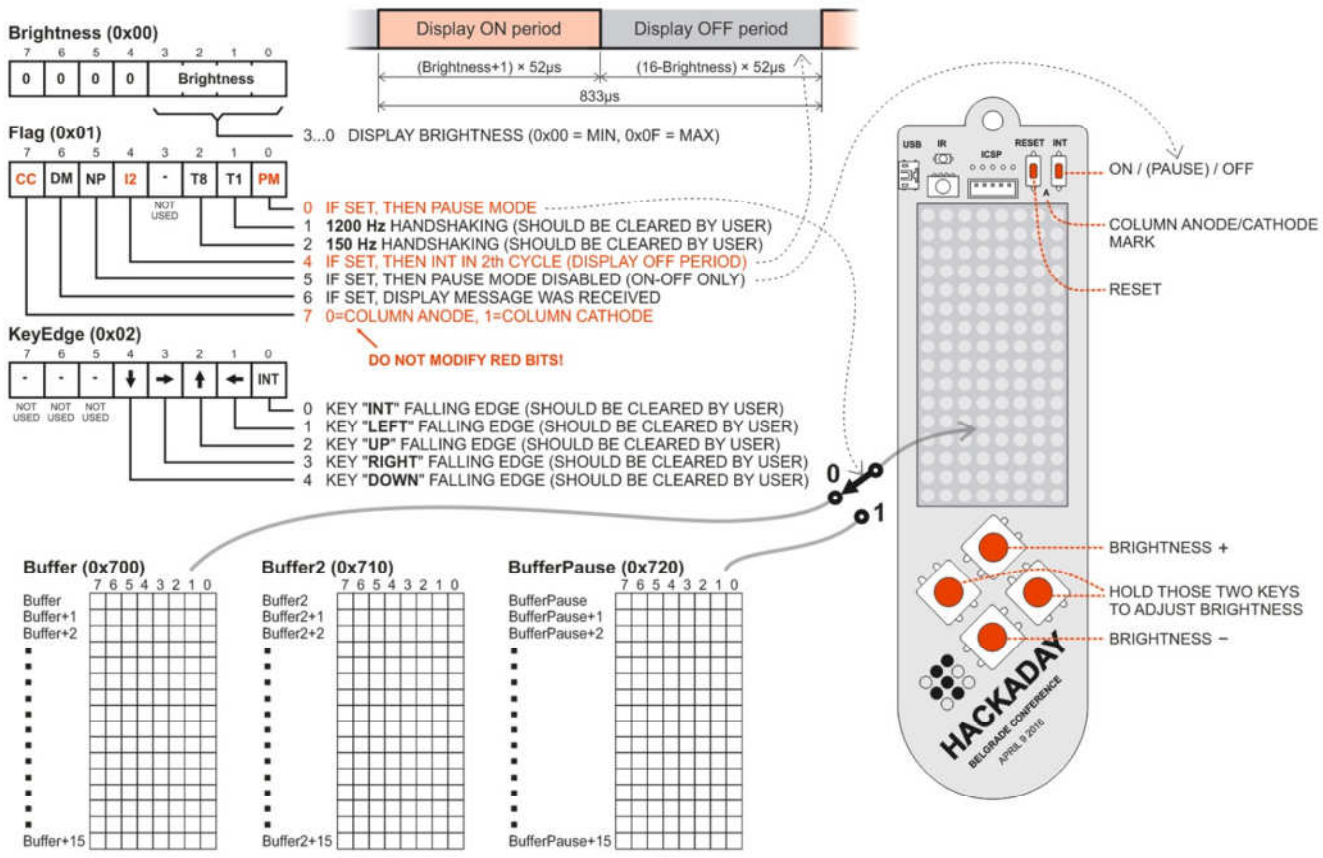
header is detected and if the serial number matches, the message will be received, otherwise it will be ignored. When the new message is received, the old one is automatically cleared.

Message terminator is pause, which is at least 200ms long. At the end of the the message, terminator 0x00 will be inserted automatically to the received string. So the maximal message is 255 bytes, although the maximal received message is 254 bytes. Header is not written, so it should not be counted in maximal message length.

At the same time, while the message is written in Data RAM, it is also written in EEPROM, at locations 0x01...0xFF.

This process of message reception is used in moving message demo application. Message can be transferred from the computer via infrared terminal and it will be received by moving message firmware and immediately displayed.

Some described aspects are represented at the following drawing:



5. DEMO APPLICATION

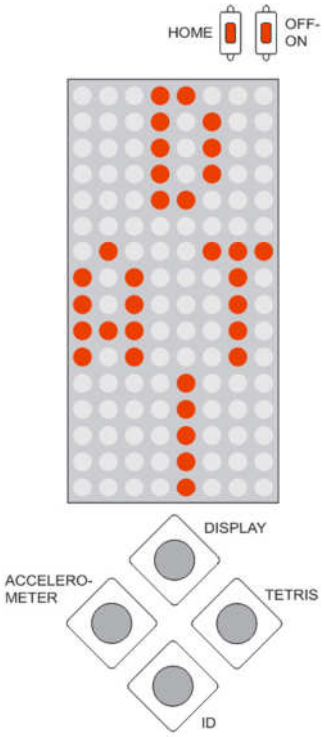
Demo firmware is linked with Kernel and loaded with Bootloader. If you wish to write your own firmware, you will probably have to dismiss the demo firmware module and link your firmware with Kernel. In that case, you will need following files from the HAD_BADGE folder:

- 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 badge displays the following Home screen, with the following button functions:



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

5.1. DISPLAY

This is the moving message display program, which runs the ASCII message located at Data RAM addresses 0x600-0x6FF. By default, there is the 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 used just in a few 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)

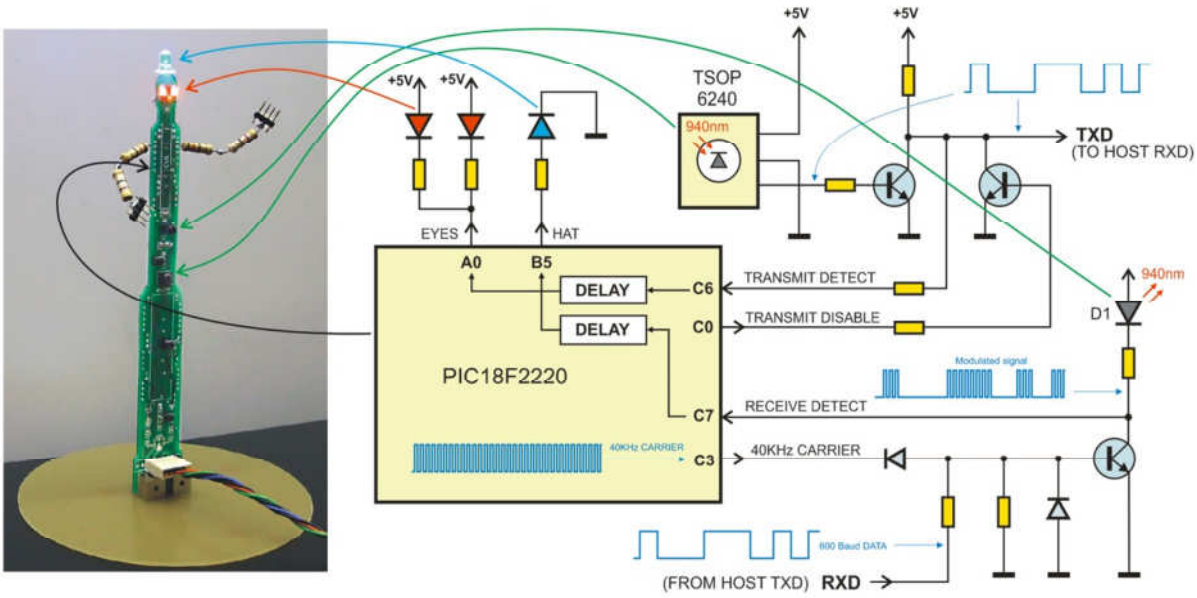
The new text can be loaded via infrared interface. During the conference, there will be the infrared terminal with the computer, which will allow typing the custom text and transmitting it to the badge. All listed commands are supported, but the message must have the header at the beginning.

The header contains 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 the new message is received, the old one is automatically cleared.

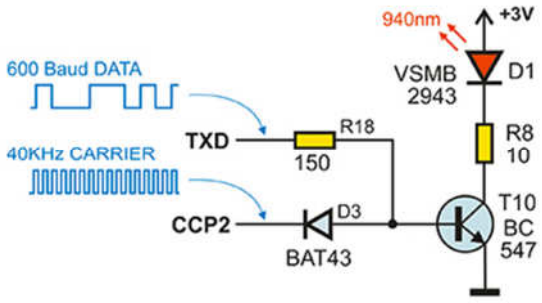
Message terminator is pause, which is at least 200ms long. That means that no special terminator is needed, and the badge will start running the message 200 ms after the whole message is transmitted. At the end of the the message, terminator 0x00 will be inserted automatically to the received string.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

The message is written in internal EEPROM, so the badge keeps it even when it is without the power supply.

Here follows the schematic diagram which explains the principle of operation of the infrared terminal:



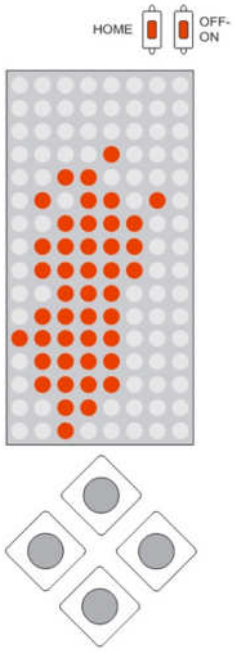
This is actually some old project which is adopted for terminal application. If you wish to build your own terminal, you can use any infrared receiver which covers 940 nm infrared range, and detects 40 KHz carrier. For the transmitter, you can use any 940 nm infrared LED, but you have to modulate the serial signal with the 40 KHz carrier. This is how it was used in the badge (note that TX signal is internally inverted in the MCU):



5.2. ACCELEROMETER

Badges are delivered without the accelerometer module, but there is the prototyping area which is adopted so that it can accept modules GY-521 (recommended) or ADXL345. If you have

one of those modules, you can solder them and use this application. Badge will automatically detect which module is present.

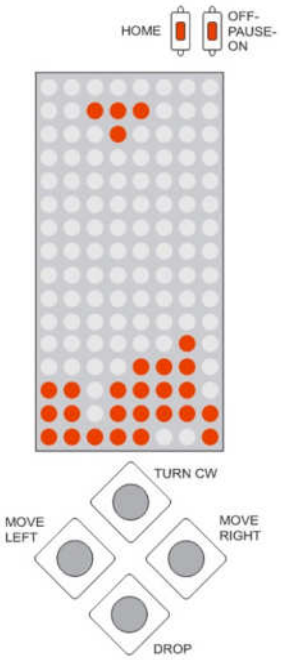


The application is very simple, it simulates the liquid (or little balls) which are moving as you tilt the badge. There is the video demonstration at
https://www.youtube.com/watch?v=a2YFsHdKY7c

5.3. 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 it, 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. Only if RESET is pressed, the game is reset also and it can be restarted from the beginning.



5.4. ID

This application doesn't have its own screen, when it is invoked by pressing the lower key, the badge transmits it unique serial number via infrared transmitter. It can be used to identify itself displaying its serial number on the infrared terminal.