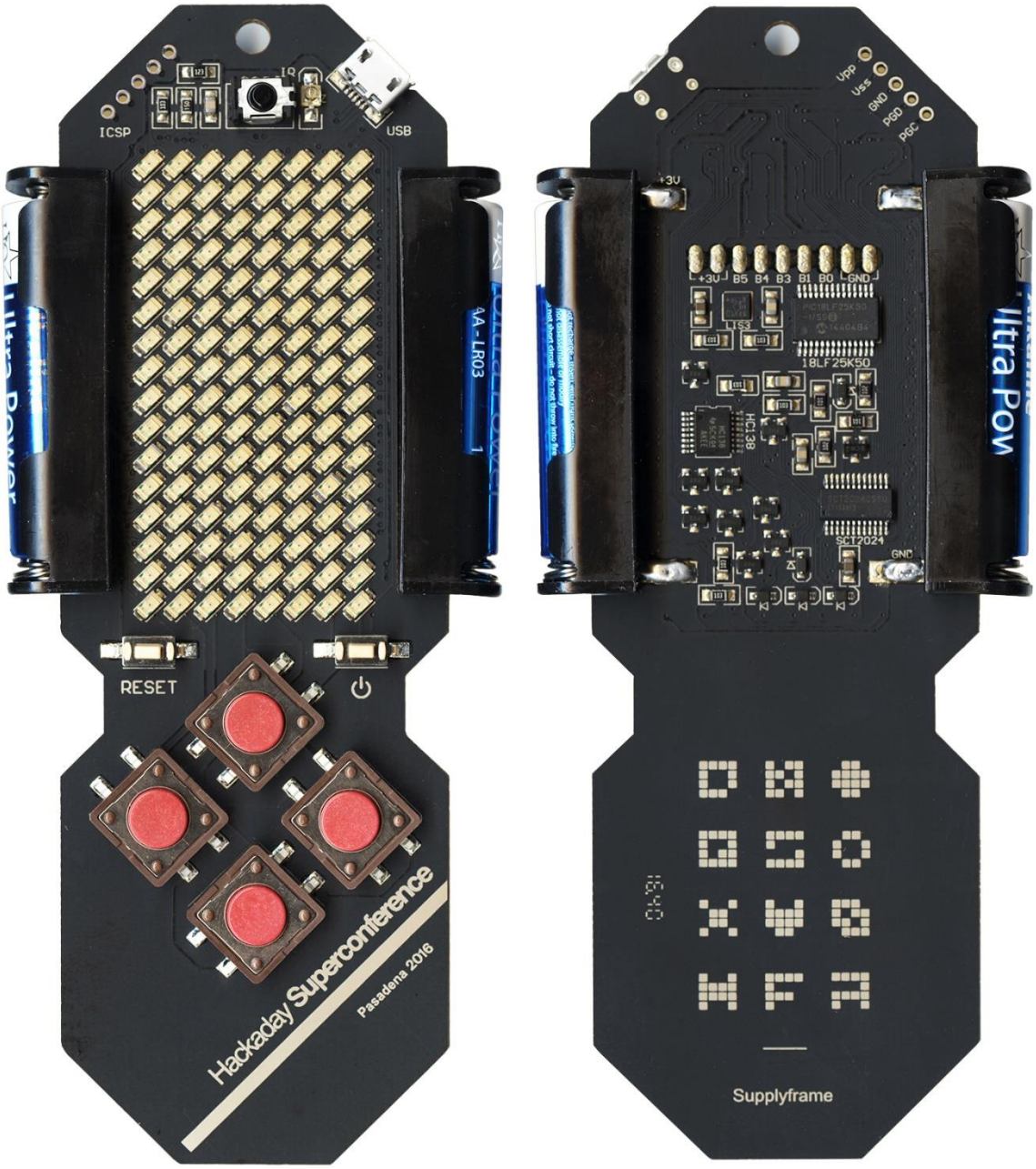
HACKADAY SUPERCONFERENCE BADGE



Pasadena, 2016 Nov. 5+6 Ver 1 Rev 0

1. INTRODUCTION

Hackaday Superconference Badge is a special goft for every conference visitor. It also acts as a hacking tool, so that everyone gets a chance to display their creativity and programming skills during the conference. Hacking will be mainly through software, but hardware hacking is also possible.

To program an 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.

To keep the badge "alive" during the conference and before the hacking event, 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, a moving message display and an accelerometer demo. 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.

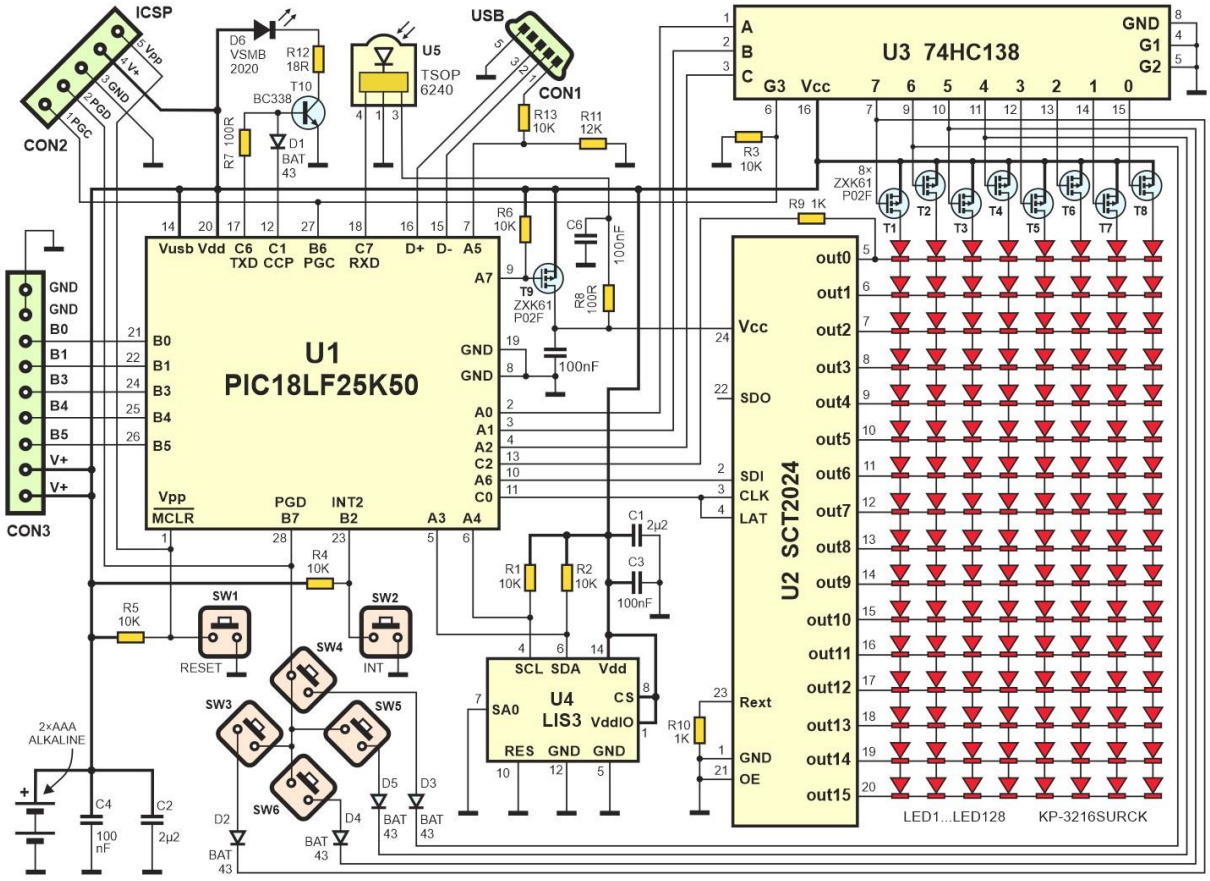
We will also be hosting a challenge event. This time, the task will consist of four independent tasks, with the special fifth task which will use all previous results to complete the final result. Every visitor is challenged to try to solve the problem during the conference, and the first one who gets the "Congratulations" message on their badge, will win a prize.

2. HARDWARE

The display consists of 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 is one USB Micro-B connector and two groups of pads, ready for connectors. One of them is 5-pole ICSP (In-Circuit Serial Programmer), and the other one is 9-pole I/O connector.

PCB dimensions are 46×139 mm. 8×16 LED matrix is built of 128 discrete SMD LEDs 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:



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 is also resident, and which may be used as a BIOS for user's application
- Demo application, which is loaded by the Bootloader, and supported by Kernel. Demo application is easily removable, so if Bootloader is used to write the new firmware, demo application will automatically be cleared from the program memory. Of course, it can programmed again using DEMO 2.HEX file

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 **RAMDEF.INC** file, and to comment out the line

```
#include <BootLoader 2b.inc>
```

which is in the **KERNEL 2.INC** 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 bootloader at MCU side, so when you connect the badge to your computer via USB port, it is recognized as the external disc drive. However, to achieve this, you must firs switch the badge to the bootloader mode. Here is how to do that:

Press both white buttons (**RESET** and **ON-OFF**)
Release the left button first, an the release the right button

At this moment, the LED in the top left corner of the badge should blink, and the new drive named **HackABadge** should appear on your computer's screen. All you have to do is to drag and drop the **HEX** file on the **HackABdge** icon. After the taskbar finishes its its job and disappears, you have to start your application on the badge, by pressing the **ON-OFF** (the right white button) on the badge, or simply disconnecting the USB cable.

2.3. MEMORY ORGANIZATION

PIC18LF25K50 contains 32 K (0x8000) Program Flash memory, 2 K (0x0800) Data RAM memory and 256 (0x0100) 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).

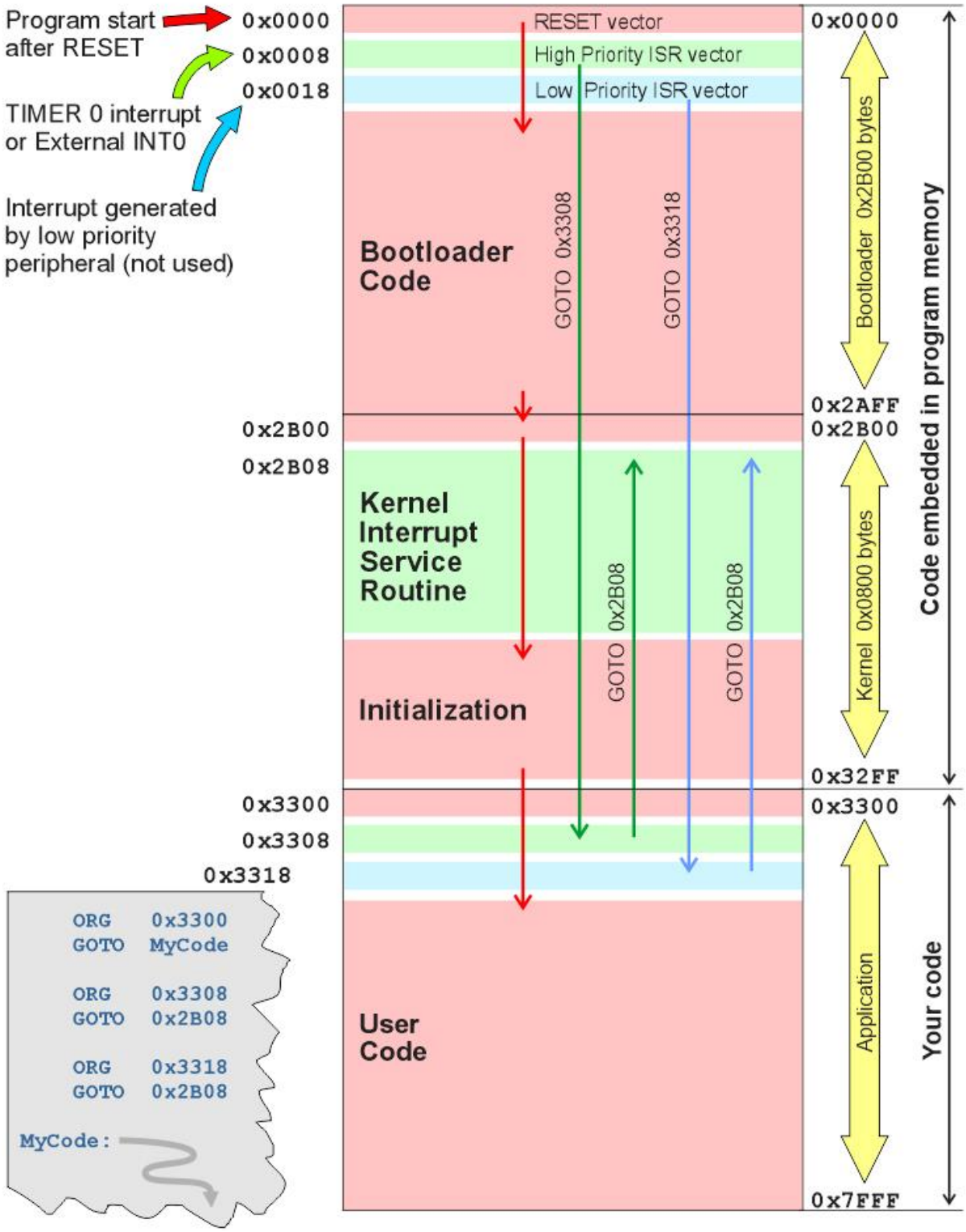
2.3.1. PROGRAM MEMORY ORGANIZATION

Bootloader and kernel are integrated and they are used as the main firmware module, taking 0x3300 bytes of MCU's program space. User application must start at 0x3300 and must contain redirection vectors for HIgh Priority ISR (Interrupt Service Routine) at the address 0x3308, and for Low Priority ISR at the address 0x3318. By default, they should contain jumps to 0x2B08, 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 are a few subroutines which are available to the user. One of them is a 32-bit pseudorandom number subroutine, with the entry point at the address 0x2B04. The execution time for this subroutine is 79 T cycles (which is 6.6 µs), 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.


Program start after RESET
TIMER 0 interrupt or External INT0
Interrupt generated by low priority peripheral (not used)
0x0000
0x0008
0x0018
RESET vector
High Priority ISR vector
Low Priority ISR vector
Bootloader Code
GOTO 0x3308
GOTO 0x3318
0x2B00
0x2B08
Kernel Interrupt Service Routine
Initialization
GOTO 0x2B08
GOTO 0x2B08
0x3300
0x3308
0x3318
User Code
ORG 0x3300
GOTO MyCode
ORG 0x3308
GOTO 0x2B08
ORG 0x3318
GOTO 0x2B08
MyCode:
0x0000
0x2AFF
0x2B00
0x32FF
0x3300
0x7FFF
Bootloader 0x2B00 bytes
Kernel 0x0800 bytes
Application
Code embedded in program memory
Your code

2.3.2. DATA RAM ORGANIZATION

Kernel uses Data RAM GPR (General Purpose Register) area locations 0x000...0x029 for housekeeping registers. Locations 0x600...0737 are used for special purpose.

addr	*name*	*bit* *description*
0x00	KeyEdge	0 = set by kernel if key INT edge detected (user must clear)
		1 = set by kernel if key LEFT edge detected (user must clear)
		2 = set by kernel if key UP edge detected (user must clear)
		3 = set by kernel if key DOWN edge detected (user must clear)
		4 = set by kernel if key RIGHT edge detected (user must clear)
		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 mode, clr by kernel: Run mode (do not modify)
		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 (set/clr by user)
		4 = Timer 0 interrupt in 2nd phase (LED OFF period) (do not modify)
		5 = set: Disable pause mode
		6 = Flag that display message was received (user must reset)
		7 = Not used
0x07	RXFlag	0 = set: Enable RX to RAM 0x601...0x6FF and EEprom (set/clr by user)
		1 = set: RX header reception is in progress (do not modify)
		2 = set: RX message reception is in progress (do not modify)
		3 = set: RX message received (internal use, do not modify it)
		4...7 Not used
0x08	Brightness	Display PWM, user presets to 0...15 for dimming
0x09	GPreg	General purpose register, may be used by user
0x0A	Anode Count	display multiplex column counter 0...7
0x0B	BitMask	10000000...00000001, 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

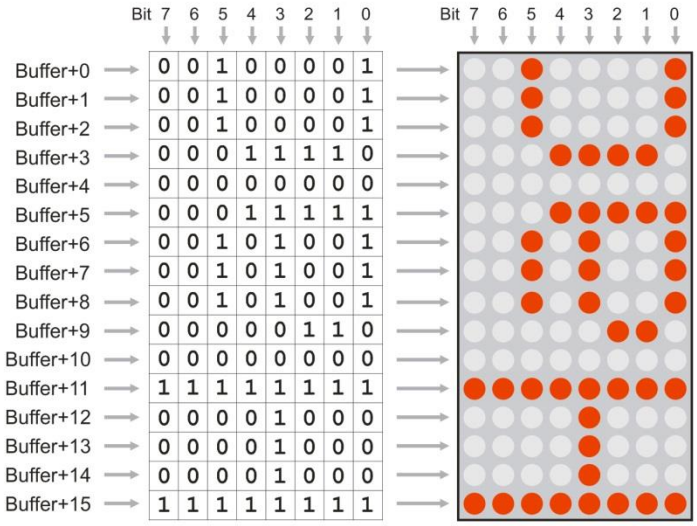
0x10	RXptr	Low RXD buffer pointer (high is always =6)
0x11	RXpatience	Patience counter, preset when byte received, count down
0x12	PowerOFF	Auto Power Off period (×6 sec), preset here to alter timing
0x13	PowerCount	Auto Power Off count down
0x14	Inner	GP register, may be used by user
0x15	Outer	GP register, may be used by user
0x16	Uniform	(2 bytes) 150 Hz divider, count up for 6 sec timing
0x18	RXserial	(2 bytes) Received serial number (binary), ready for comparison
0x1A	MySerial	(2 bytes) unit serial number copied from ROM address 0x100E
0x1C	FSR0temp	(2 bytes) Temporary FSR0 during INT
0x1E	AccX	; Accelerometer X data (Little Endian, Left justified)
0x20	AccY	; Accelerometer Y data (Little Endian, Left justified)
0x22	AccZ	; Accelerometer Z data (Little Endian, Left justified)

0x1E...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 (don't modify)
0x734...0x737	RND internal arithmetic registers (may be used for another purpose)
0x738...0x7FF	User data RAM space

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 (0x601-0x60E), 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/pause 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. 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.

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.

At last, it enables TIMER 0 interrupt and jumps to user's program at 0x3300.

Port C2 (which is used by Bootloader to blink one LED) 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 the allowed range.

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 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 and increase its entropy, so W 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 0x2B08 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 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 (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.

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) so one of bits 0...4 in register KeyEdge will be set. 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 (it 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 only 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 a 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,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. So it slleps inside the interrupt routine.

When the INT key is pressed again, 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 254 bytes will be detected, and the rest will be ignored.

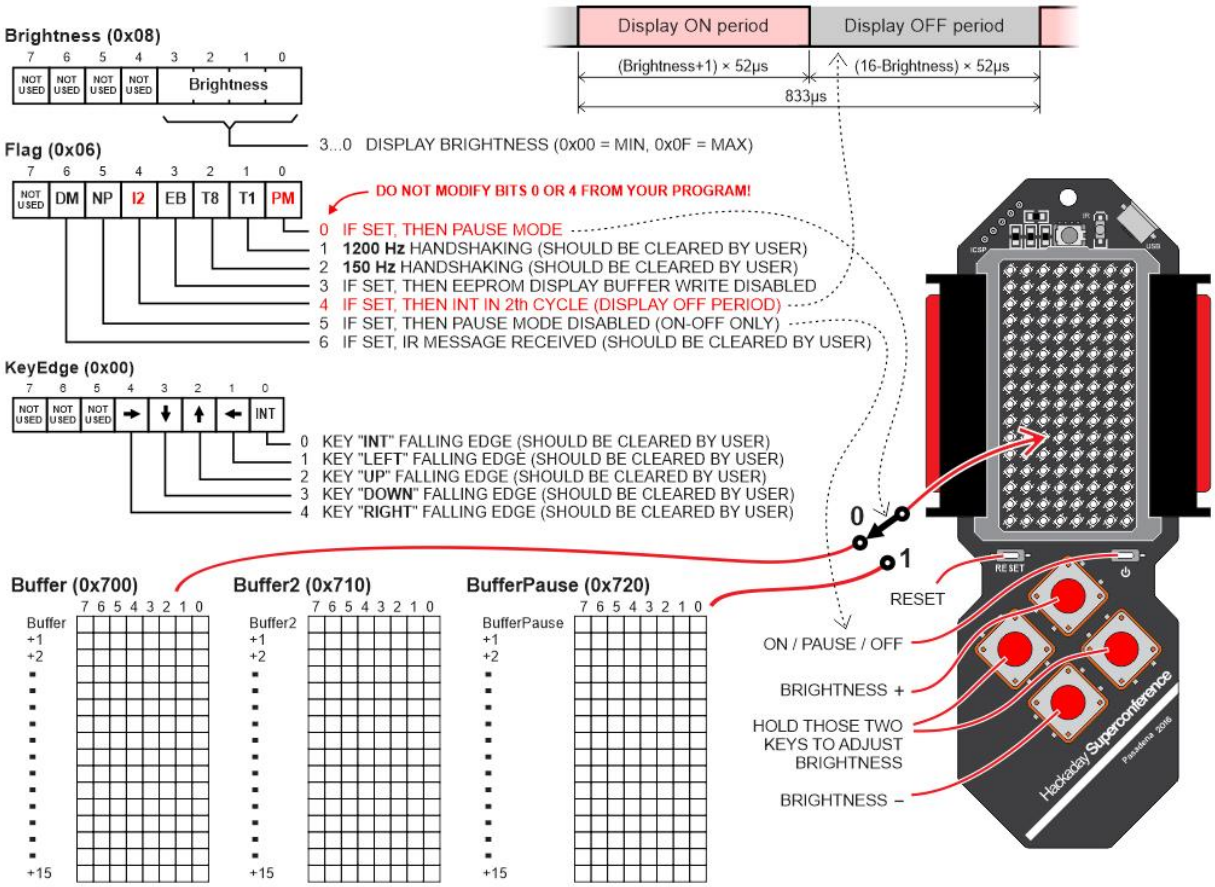
Message header contains ASCII "[", one to five ASCII digits (which represent recipient's serial number in decimal form) and ASCII"]". So, header can vary from [0] to [65535]. Only if RXFlag,0 is set, valid header is detected and if the serial number matches, the message will be received, otherwise it 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 in data memory. Thus, 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 (EEPROM location 0x00 is used for brightness setting).

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

Some aspects described here are represented on the following drawing:


Brightness (0x08)
7 6 5 4 3 2 1 0
NOT USED
NOT USED
NOT USED
NOT USED
Brightness
3...0 DISPLAY BRIGHTNESS (0x00 = MIN, 0x0F = MAX)
Display ON period
Display OFF period
(Brightness+1) × 52µs
(16-Brightness) × 52µs
833µs
Flag (0x06)
7 6 5 4 3 2 1 0
NOT USED
DM
NP
I2
EB
T8
T1
PM
DO NOT MODIFY BITS 0 OR 4 FROM YOUR PROGRAM!
0 IF SET, THEN PAUSE MODE
1 1200 Hz HANDSHAKING (SHOULD BE CLEARED BY USER)
2 150 Hz HANDSHAKING (SHOULD BE CLEARED BY USER)
3 IF SET, THEN EEPROM DISPLAY BUFFER WRITE DISABLED
4 IF SET, THEN INT IN 2th CYCLE (DISPLAY OFF PERIOD)
5 IF SET, THEN PAUSE MODE DISABLED (ON-OFF ONLY)
6 IF SET, IR MESSAGE RECEIVED (SHOULD BE CLEARED BY USER)
KeyEdge (0x00)
7 6 5 4 3 2 1 0
NOT USED
NOT USED
NOT USED
INT
0 KEY "INT" FALLING EDGE (SHOULD BE CLEARED BY USER)
1 KEY "LEFT" FALLING EDGE (SHOULD BE CLEARED BY USER)
2 KEY "UP" FALLING EDGE (SHOULD BE CLEARED BY USER)
3 KEY "DOWN" FALLING EDGE (SHOULD BE CLEARED BY USER)
4 KEY "RIGHT" FALLING EDGE (SHOULD BE CLEARED BY USER)
Buffer (0x700)
Buffer2 (0x710)
BufferPause (0x720)
7 6 5 4 3 2 1 0
Buffer
Buffer2
BufferPause
+1
+2
+15
0
1
RESET
ON / PAUSE / OFF
BRIGHTNESS +
HOLD THOSE TWO KEYS TO ADJUST BRIGHTNESS
BRIGHTNESS –
RESET
Hackaday Superconference
Pasadena 2016