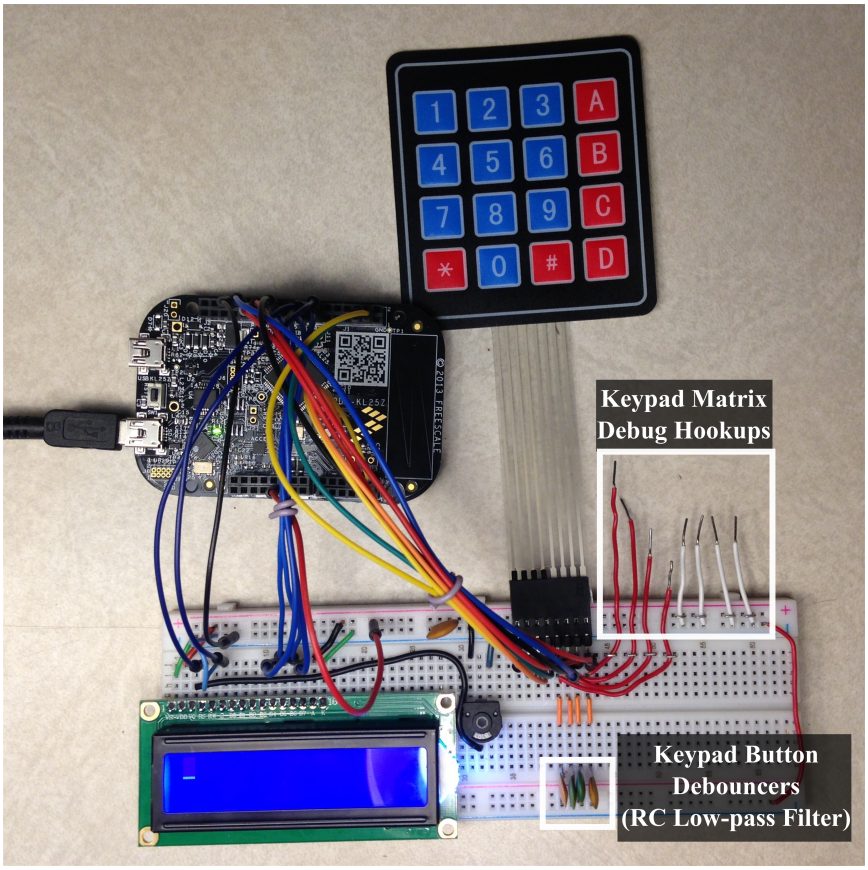
Lab 4: UART & GPIO Interrupts

Lewis Collum, Tanner Wilson, Rayman Alli

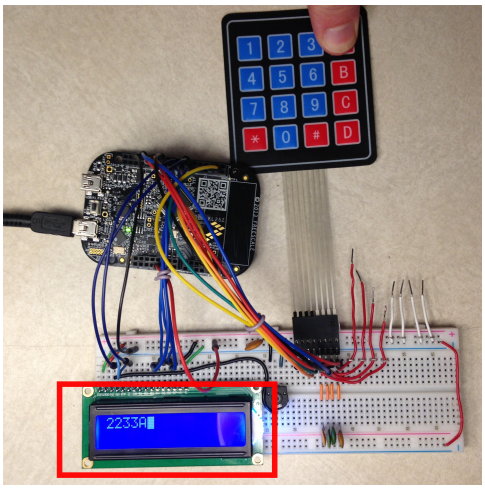
April 2, 2019

1 PART A

1.1 Setup

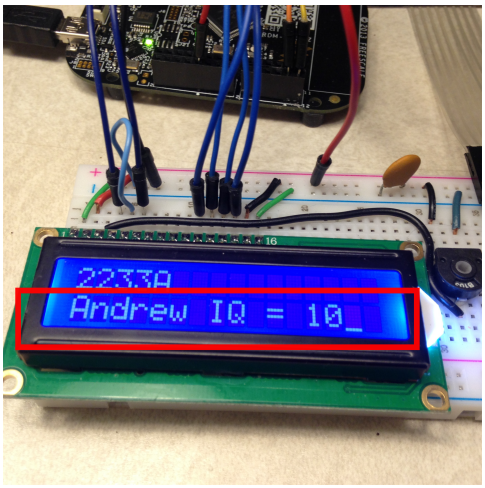


1.2 Sending to UART



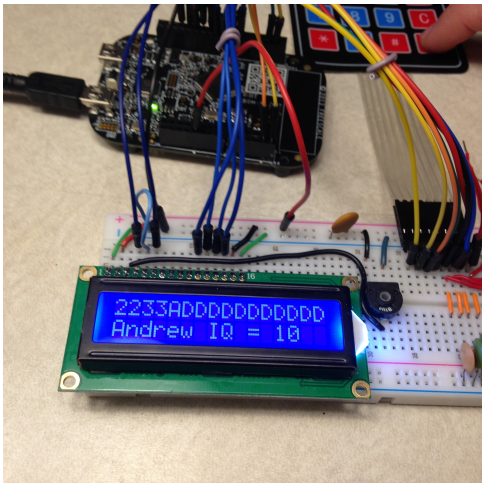

```
[charon@pluto ~]$ sudo picocom -b 115200 /dev/ttyACM0
picocom v3.1

port is        : /dev/ttyACM0
flowcontrol    : none
baudrate is    : 115200
parity is      : none
databits are   : 8
stopbits are   : 1
escape is      : C-a
local echo is  : no
noinit is      : no
noreset is     : no
hangup is      : no
nolock is      : no
send_cmd is    : sz -vv
receive_cmd is : rz -vv -E
imap is        :
omap is        :
emap is        : crcrlf,delbs,
logfile is     : none
initstring     : none
exit_after is  : not set
exit is        : no

Type [C-a] [C-h] to see available commands
Terminal ready
2233A
```

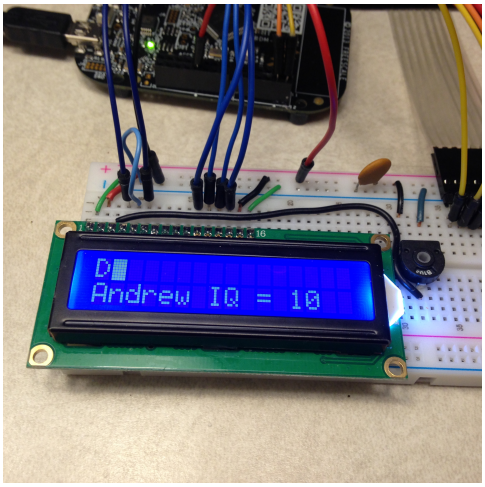
1.3 Fetching from UART



1.4 Clearing

Once a line on the LCD exceeds the length of a user-defined value (in this case 16), the LCD will clear the line.





1.5 Code

The main file is below. All other code, including "uart", "lcd", and "keypad" can be found on GitHub at `https://github.com/LewisCollum/MKL25Z4_Interrupts`. The files of interest are in the "source" directory.

Importantly, the control flow results from a state machine where the interrupts change the state. This can be seen in the "run" function.

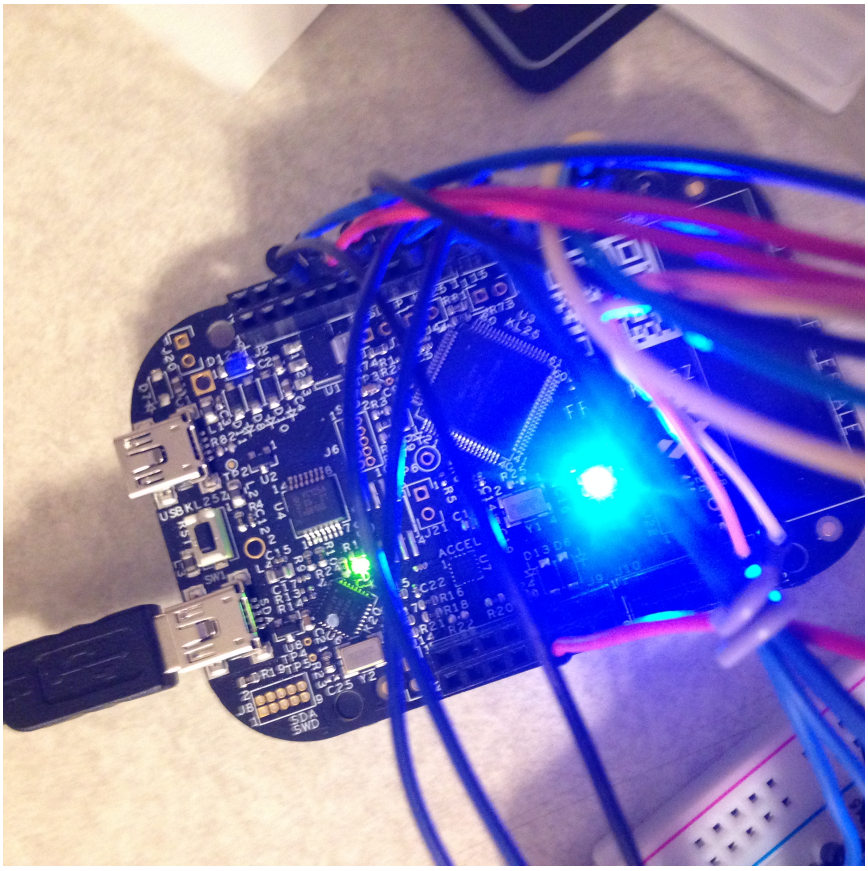
```
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
```

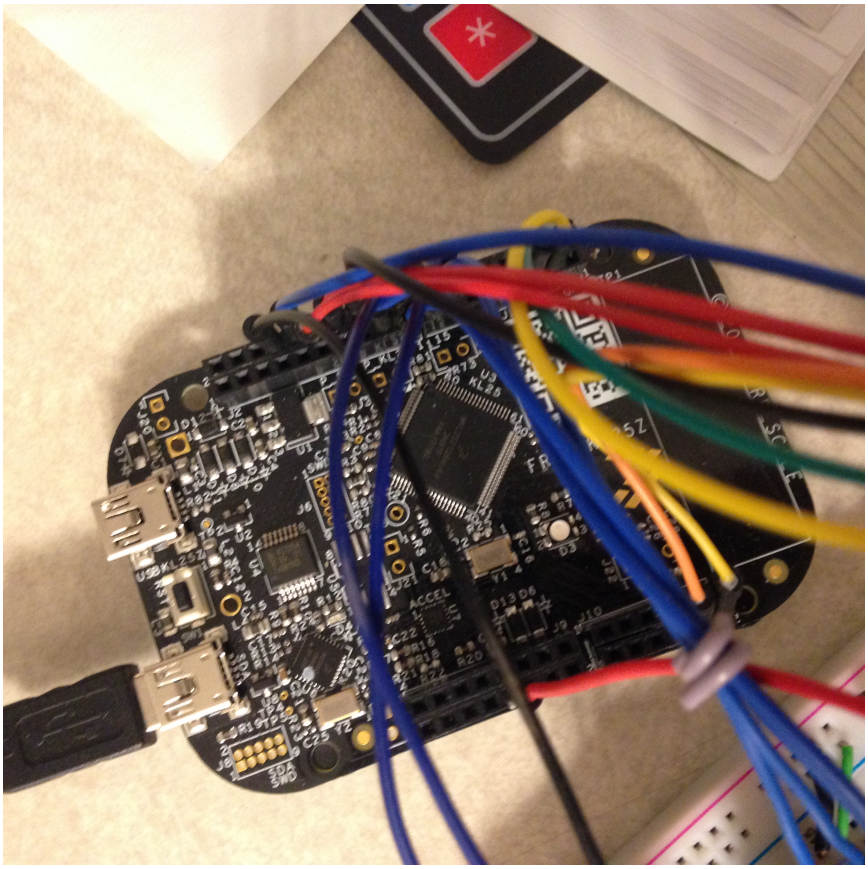
```
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "delay.h"
#include "keypad.h"
#include "lcd.h"
#include "common.h"
#include "uart.h"

unsigned char key;
unsigned char uartKey;

enum RunStates {waiting, sending, fetching} state;

void setup() {
  __disable_irq();

  BOARD_InitBootPins();
  BOARD_InitBootClocks();
  BOARD_InitBootPeripherals();
  BOARD_InitDebugConsole();

  state = waiting;
  __enable_irq();
}

void run() {
  while (state == waiting);

  __disable_irq();
  if (state == sending) {
        lcdWriteDataToRow(key, 0);
        uartSend(key);
        while(uartIsSending());
        state = waiting;
  }
  else if (state == fetching) {
        lcdWriteDataToRow(uartKey, 1);
        state = waiting;
  }
```

```
  __enable_irq();
}

int main(void) {
  setup();
  while(1) run();
  return 0;
}

void PORTD_IRQHandler() {
  if(state == waiting) {
        key = keypadGetPressedKey();
        state = sending;
  }
  PORTD->ISFR = 0xF;
}

void UART0_IRQHandler() {
  if(state == waiting) {
        uartKey = UART0->D;
        state = fetching;
  }
}
```

2 PART B

2.1 Blinking Blue LED due to Timer Interrupt





2.2 Code

```
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"

#define clockPortD 0x1000
#define clockTMP0 0x01000000
#define gpio 0x100

int main(void) {
```

```
        __disable_irq();
        BOARD_InitBootClocks();

        SIM->SCGC5 |= clockPortD;
        PORTD->PCR[1] = gpio;
        PTD->PDDR |= 0b10;

        SIM->SOPT2 |= 0x01000000;           /* use MCGFLLCLK as timer counter clock */

        SIM->SCGC6 |= clockTMP0;           /* enable clock to TPM0 */
        TPM0->SC = 0;                       /* disable timer while configuring */
        TPM0->SC = 0x07;                    /* prescaler /128 */
        TPM0->MOD = 0xFFFF; // max modulo value
        TPM0->SC |= 0x80; // clear TOF
        TPM0->SC |= 0x40; // enable timeout interrupt
        TPM0->SC |= 0x08; // enable timer
        NVIC->ISER[0] |= 0x20000; // enable IRQ17

        __enable_irq();

        while (1);
        return 0;
}

void TPM0_IRQHandler() {
        PTD->PTOR = 0x02; // toggle Blue LED
        TPM0->SC |= 0x80; // clear TOF
}
```