LPC2138 4×4 Keypad Calculator — Proteus Simulation + Git Repository Pack

This package gives you everything you need to simulate a simple calculator on an **NXP LPC2138 (ARM7TDMI-S)** in **Proteus**, plus a ready-to-use **Git repo structure and README**.

---

1) What you'll build

A 16×2 LCD + 4×4 matrix keypad calculator that performs **+, −, ×,** ÷ with up to 8-digit integers. It runs on LPC2138 and is fully testable in Proteus.

---

2) Requirements

Software

- Proteus (8.x or newer with ARM7 library that includes LPC2138)
- Keil µVision (ARM) or GCC/Make (Windows or Linux)
- Optional: VS Code + Arm-None-EABI toolchain

Hardware models in Proteus

- LPC2138 MCU
- 16×2 LCD (HD44780 compatible)
- 4×4 Matrix Keypad
- 10 kΩ resistors (pull-ups for keypad columns)
- 12 MHz crystal + 22 pF caps (or use internal clock model if supported by your Proteus part)

---

3) Pin mapping (recommended)

LCD (4-bit mode)

- RS → P0.0
- E → P0.1
- D4 → P0.4
- D5 → P0.5
- D6 → P0.6
- D7 → P0.7
- RW → GND

Keypad 4×4

- Rows (R0..R3) → P0.16, P0.17, P0.18, P0.19 (outputs)
- Cols (C0..C3) → P0.20, P0.21, P0.22, P0.23 (inputs with 10 kΩ pull-ups to VCC)

Keep all these pins in **GPIO** mode by ensuring `PINSEL0` selects function **00** for the above P0 pins.

Clock/Reset

- Use 12 MHz crystal on LPC2138; you can run without PLL in simulation to keep things simple.

4) Schematic wiring checklist (Proteus)

1. Place **LPC2138** and set **Clock** to 12 MHz.
2. Place **LM016L/Generic 16×2 LCD**. Wire RS/E/D4..D7 as in the map. Tie **RW to GND**.
3. Place **Matrix Keypad**. Connect rows to P0.16..P0.19 and columns to P0.20..P0.23.
4. Add **pull-up resistors (10 kΩ)** from each column line to VCC.
5. Add **Power** (VCC/5V) and **Ground** rails. (HD44780 LCD is 5 V tolerant; in simulation you can power MCU at 3.3 V equivalent. If mixing 5 V with 3.3 V, level-shifting is recommended in hardware; Proteus simulation is tolerant.)
6. Place **Reset button** if you want manual reset (tie RESET pin via pull-up and momentary to GND).
7. Double-check no pins are floating.

5) Firmware (Keil/GCC) — `src/main.c`

Minimal, no-PLL build intended for simulation. Uses software delays for LCD timing and debounced keypad scanning.

```c
#include <LPC213x.h>  // Works for LPC2138 in Keil; for GCC supply proper header

// ===== Pin Defines (match the pin map above) =====
#define LCD_RS (1u<<0)   // P0.0
#define LCD_E  (1u<<1)   // P0.1
#define LCD_D4 (1u<<4)   // P0.4
#define LCD_D5 (1u<<5)   // P0.5
#define LCD_D6 (1u<<6)   // P0.6
#define LCD_D7 (1u<<7)   // P0.7

#define KP_R0  (1u<<16)  // rows as outputs
#define KP_R1  (1u<<17)
#define KP_R2  (1u<<18)
#define KP_R3  (1u<<19)
```

```
#define KP_C0  (1u<<20)  // cols as inputs (with external pull-ups)
#define KP_C1  (1u<<21)
#define KP_C2  (1u<<22)
#define KP_C3  (1u<<23)

// Convenience masks
#define LCD_PINS (LCD_RS|LCD_E|LCD_D4|LCD_D5|LCD_D6|LCD_D7)
#define KP_ROWS  (KP_R0|KP_R1|KP_R2|KP_R3)
#define KP_COLS  (KP_C0|KP_C1|KP_C2|KP_C3)

// ===== Simple delay (simulation) =====
static void delay_cycles(volatile unsigned int c){ while(c--) __asm__
volatile("nop"); }
static void ms(unsigned int t){ while(t--) delay_cycles(8000); } // ~approx at
12 MHz

// ===== LCD Low-level =====
static void lcd_pulse_e(){ IOSET0 = LCD_E; delay_cycles(200); IOCLR0 = LCD_E; }
static void lcd_write4(unsigned char nibble){
    // Clear D4..D7 then set according to nibble
    IOCLR0 = (LCD_D4|LCD_D5|LCD_D6|LCD_D7);
    if(nibble & 0x01) IOSET0 = LCD_D4;
    if(nibble & 0x02) IOSET0 = LCD_D5;
    if(nibble & 0x04) IOSET0 = LCD_D6;
    if(nibble & 0x08) IOSET0 = LCD_D7;
    lcd_pulse_e();
}
static void lcd_cmd(unsigned char cmd){
    IOCLR0 = LCD_RS;              // RS=0 for command
    lcd_write4(cmd>>4);
    lcd_write4(cmd & 0x0F);
    ms(2);
}
static void lcd_data(unsigned char data){
    IOSET0 = LCD_RS;              // RS=1 for data
    lcd_write4(data>>4);
    lcd_write4(data & 0x0F);
    ms(2);
}
static void lcd_init(){
    // Configure GPIO
    PINSEL0 &= ~(0x3FFFFF); // ensure P0.0..P0.21 are GPIO where used
    IODIR0  |= (LCD_PINS | KP_ROWS); // LCD pins + keypad rows as outputs
    IODIR0  &= ~(KP_COLS);           // keypad cols as inputs

    // LCD init sequence (4-bit)
    ms(40);
    IOCLR0 = LCD_RS;
    lcd_write4(0x03); ms(5);
    lcd_write4(0x03); ms(5);
    lcd_write4(0x03); ms(1);
    lcd_write4(0x02); // 4-bit mode
    lcd_cmd(0x28);    // 2 lines, 5x8 font
    lcd_cmd(0x0C);    // display ON, cursor OFF
    lcd_cmd(0x06);    // entry mode
    lcd_cmd(0x01);    // clear
    ms(2);
}
static void lcd_gotoxy(unsigned char x, unsigned char y){
    unsigned char addr = (y==0)? 0x00 : 0x40;
    addr += x;
    lcd_cmd(0x80 | addr);
}
static void lcd_print(const char* s){ while(*s) lcd_data(*s++); }
static void lcd_print_num(long val){
    char buf[16];
    int i=0, j, neg=0;
    if(val==0){ lcd_data('0'); return; }
    if(val<0){ neg=1; val=-val; }
    while(val>0 && i<15){ buf[i++] = '0' + (val%10); val/=10; }
    if(neg) buf[i++]='-';
    for(j=i-1;j>=0;j--) lcd_data(buf[j]);
}

// ===== Keypad scan =====
// Key layout (example):
// [ 1 ] [ 2 ] [ 3 ] [ + ]
// [ 4 ] [ 5 ] [ 6 ] [ - ]
// [ 7 ] [ 8 ] [ 9 ] [ * ]
// [ C ] [ 0 ] [ = ] [ / ]

static const char keymap[4][4] = {
    {'1','2','3','+'},
    {'4','5','6','-'},
    {'7','8','9','*'},
    {'C','0','=', '/'}
};

static int read_cols(){ return (IOPIN0 & KP_COLS); }
static void drive_row(int r){
    // Set all rows high, then pull selected row low
    IOSET0 = KP_ROWS;
    switch(r){
        case 0: IOCLR0 = KP_R0; break;
        case 1: IOCLR0 = KP_R1; break;
        case 2: IOCLR0 = KP_R2; break;
        case 3: IOCLR0 = KP_R3; break;
    }
}

static char keypad_getkey(){
    for(int r=0;r<4;r++){
        drive_row(r);
        delay_cycles(1000);
        int cols = read_cols();
        // Because of pull-ups, pressed column will read LOW
        if(!(cols & KP_C0)) { while(!(read_cols() & KP_C0)); ms(10); return
keymap[r][0]; }
        if(!(cols & KP_C1)) { while(!(read_cols() & KP_C1)); ms(10); return
keymap[r][1]; }
        if(!(cols & KP_C2)) { while(!(read_cols() & KP_C2)); ms(10); return
keymap[r][2]; }
        if(!(cols & KP_C3)) { while(!(read_cols() & KP_C3)); ms(10); return
keymap[r][3]; }
    }
    return 0; // no key
}

// ===== Calculator state =====
enum {S_FIRST=0, S_OP, S_SECOND, S_SHOW} state = S_FIRST;
long op1=0, op2=0; char op='\0'; int digits=0;

static void reset_calc(){ op1=op2=0; op='\0'; digits=0; state=S_FIRST;
lcd_cmd(0x01); ms(2); lcd_print("Calc LPC2138"); lcd_gotoxy(0,1); }

static void append_digit(long *target, int d){ if(digits<8){ *target =
(*target)*10 + d; digits++; } }

static long compute(long a, long b, char o, int* err){
    *err = 0;
    switch(o){
        case '+': return a + b;
        case '-': return a - b;
        case '*': return a * b;
        case '/': if(b==0){ *err=1; return 0; } return a / b;
        default: return 0;
    }
}

int main(void){
    lcd_init();
    reset_calc();

    while(1){
        char k = keypad_getkey();
        if(!k){ continue; }
        if(k=='C'){ reset_calc(); continue; }

        if(state==S_FIRST){
            if(k>='0'&&k<='9'){ append_digit(&op1, k-'0'); lcd_data(k); }
            else if(k=='+'||k=='-'||k=='*'||k=='/'){
                op = k; state = S_OP; digits=0; lcd_data(' '); lcd_data(k);
lcd_data(' ');
            }
        }
        else if(state==S_OP){
            if(k>='0'&&k<='9'){ state=S_SECOND; append_digit(&op2, k-'0');
lcd_data(k); }
        }
        else if(state==S_SECOND){
            if(k>='0'&&k<='9'){ append_digit(&op2, k-'0'); lcd_data(k); }
            else if(k=='='){
                int err=0; long res = compute(op1, op2, op, &err);
                lcd_gotoxy(0,1);
                if(err){ lcd_print("ERR: DIV0   "); }
                else{ lcd_print("= "); lcd_print_num(res); lcd_print("
"); }
                state=S_SHOW;
            }
        }
        else if(state==S_SHOW){
            // After showing result, next digit starts new calc
            if(k>='0'&&k<='9'){ reset_calc(); append_digit(&op1, k-'0');
lcd_data(k); }
            else if(k=='C'){ reset_calc(); }
        }
    }
}
```

Notes

- For GCC, replace the header include with the correct `LPC2138.h` and provide a linker script.
- `PINSEL0` mask is coarse here; if you add peripherals, mask only the specific bits you use.
- Uses crude delays adequate for simulation; replace with timer-based delays for production.

6) Build & load

Keil µVision

1. New µVision Project → Select **NXP LPC2138**.
2. Add `src/main.c`. Set Target → Xtal = **12.0 MHz**. Disable PLL in startup if present.
3. Project → Options for Target → Output → **Create HEX File**.
4. Build → get `Objects/your_target.hex`.

Proteus

1. Double-click LPC2138 → Program File → select the **.hex**.
2. Set **Clock** to **12 MHz**.
3. Run simulation; LCD should show `Calc LPC2138`. Press keypad to verify.

---

7) Test plan

- `12 + 34 = 46` (basic addition)
- `99 - 100 = -1` (negative result)
- `123 * 4 = 492`
- `7 / 0` → shows **ERR: DIV0**
- `C` at any time → clears and returns to idle

---

8) Common Proteus pitfalls

- **No LCD text** → Check RW=GND, contrast (add 10 kΩ pot from VO to GND/VCC), timing delays.
- **Keypad not detected** → Ensure column pull-ups and row drive logic; in Proteus keypad element, verify row/col orientation.
- **Wrong pins** → Confirm `PINSEL0` keeps pins as GPIO (function `00`).
- **Clock mismatch** → If you change frequency, delays change; LCD may misbehave.

---

9) Git repository template

```
lpc2138-calculator/
├─ src/
│  └─ main.c
├─ proteus/
│  ├─ lpc2138_calc.pdsprj        # Proteus project (create in Proteus)
│  └─ lpc2138_calc.pdsprj.LCK    # (auto-generated; do not commit)
├─ toolchain/
│  ├─ keil_project.uvprojx       # Optional: Keil project file
│   └─ gcc_linker.ld              # Optional: GCC linker script
├─ .gitignore
├─ LICENSE
└─ README.md
```

`` (Keil + generic C)

```
# Keil / uVision
*.uvoptx
*.uvguix
*.bak
*.dep
Objects/
Listings/
# Build output
*.hex
*.elf
*.map
*.axf
# Proteus lock/temp
*.LCK
*.TMP
.DS_Store
Thumbs.db
```

LICENSE (MIT)

```
MIT License

Copyright (c) 2025 <Your Name>

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Soft
```