Chapter 3

Interfacing to the ST7565 GLCD

WARNING! Ensure the jumper at J13 on the DEVKIT-S12XE development board is set to 3.3V. Otherwise the ST7565 GLCD will be permanently damaged.

Figure 3.1 shows the wiring diagram for interfacing the DEVKIT-S12XE development board to the ST7565 GLCD (without a backlight). Refer to the DEVKIT-S12XE Quick Start Guide for the location of the pins on the development board and use the male-female jumper cables to connect the DEVKIT-S12XE to the ST7565 GLCD.

The display is divided into four sections:

1. The status line
2. The stack
3. The command line
4. The menu

3.1 Using simulated SPI

We will display the status line using simulated SPI. Create a new project, copy and paste the code from GLCD1.asm, assemble the file, download and run. The display should appear as in Figure 3.2.

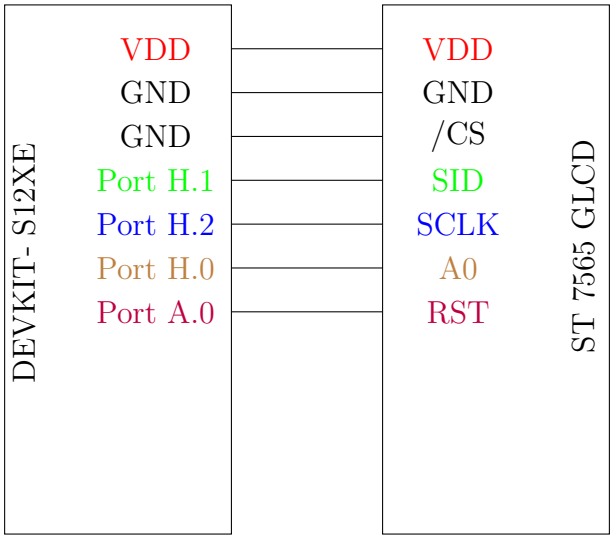

Figure 3.1: Interfacing the DEVKIT-S12XE to the ST7565 GLCD

Figure 3.2: The status line

3.2 Using SPI

We will know use SPI to display the status line, stack level labels and menu. Create a new project, copy and paste the code from GLCD2.asm, assemble the file, download and run. The display should appear as in Figure 3.3.

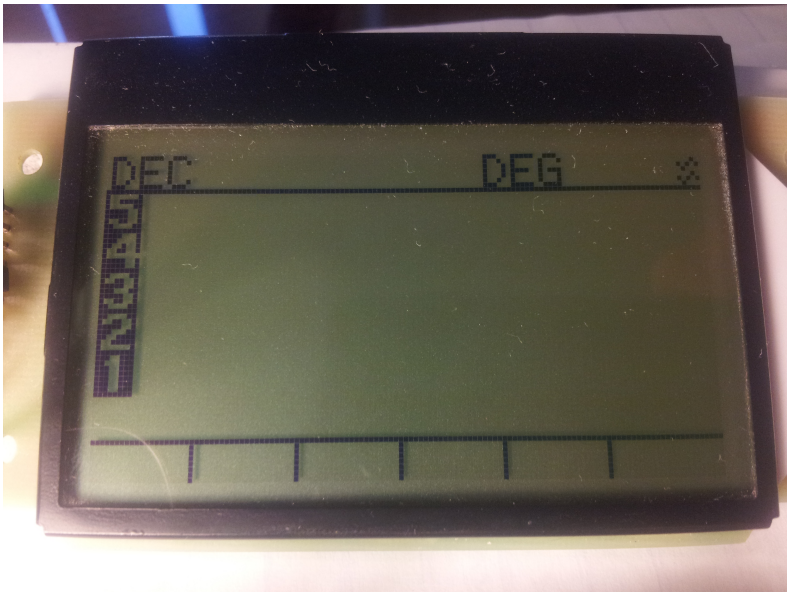

Figure 3.3: The status line, stack level labels and menu (currently empty)

3.3 Using interrupts

Finally, we add a cursor to the command line by including an interrupt in GLCD3.asm (Figure 3.4).

Figure 3.4: The completed display

In the next chapter we will complete our development system by interfacing to a matrix keypad.

Chapter 4

Interfacing to a matrix keypad

4.1 4x4 matrix keypad

Figure 4.1 shows the key numbers on the 4x4 matrix keypad, together with the row and column pins for a typical 4x4 matrix keypad (refer to your keypad's datasheet).

Figure 4.2 shows the wiring diagram for interfacing the DEVKIT-S12XE development board to the 4x4 matrix keypad. Refer to the DEVKIT-S12XE Quick Start Guide for the location of the pins on the development board and use the male-female jumper cables to connect the DEVKIT-S12XE to the 4x4 matrix keypad.

Program KYPD1.asm displays the number of the key pressed. Pressing the key in the bottom right-hand corner of the keypad will cause "00" to be displayed (Figure 4.3).

Figure 4.3: Displaying the number of a depressed key

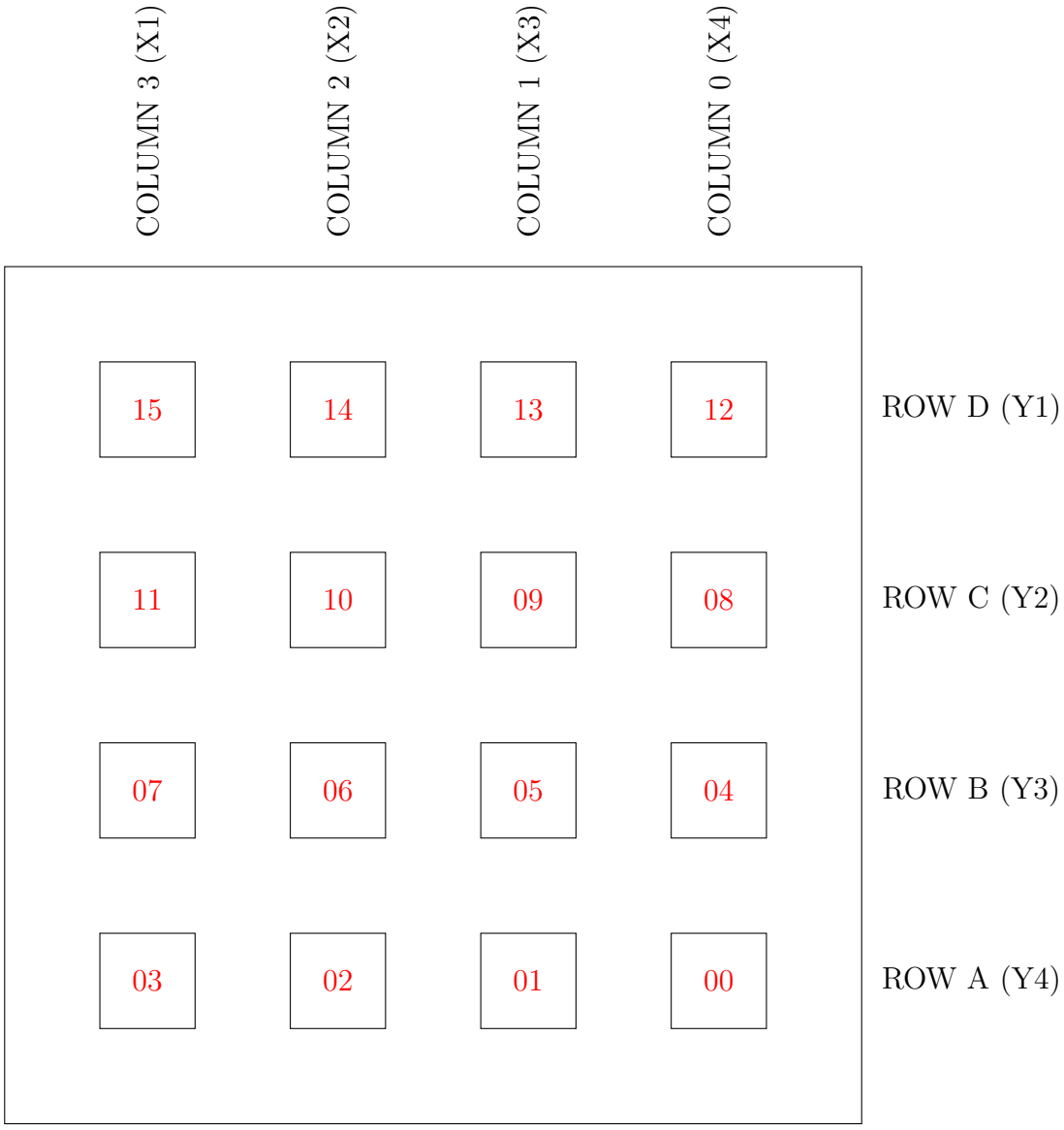

Figure 4.1: The 4x4 matrix keypad

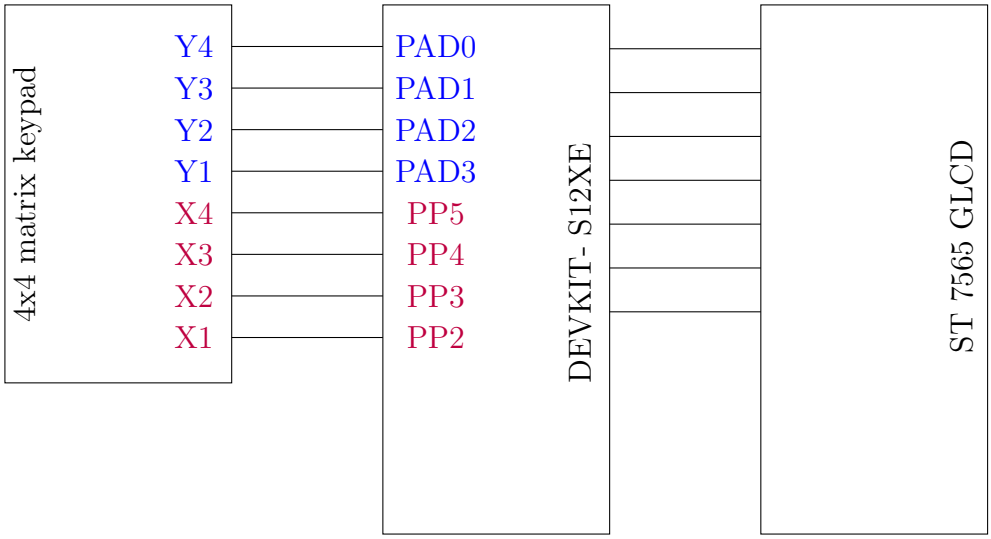

Figure 4.2: Interfacing the DEVKIT-S12XE to the 4x4 matrix keypad

4.2 8x6 matrix keypad

Figure 4.4 shows the key numbers on the 8x6 matrix keypad, together with the row and column pin names.

Figure 4.5 shows the wiring diagram for interfacing the DEVKIT-S12XE development board to the 8x6 matrix keypad. Refer to the DEVKIT-S12XE Quick Start Guide for the location of the pins on the development board and use the male-female jumper cables to connect the DEVKIT-S12XE to the 8x6 matrix keypad.

Program KYPD2.asm displays the number of the key pressed. Pressing the key in the bottom right-hand corner of the keypad will cause "00" to be displayed (Figure 4.3).

4.3 Full Chip Simulation

Program KYPD1_FCS.asm is the Full Chip Simulation (FCS) version of program KYPD1.asm. FCS enables us to debug code without having to download it to the DEVKIT-S12XE. To use FCS, select "Full Chip Simulation" instead of "P&E USB BDM Multilink" when creating the project (see Chapter 2 - "Blinking an LED"). After pasting the code from KYPD1_FCS.asm press "Make" in the tool bar and then press "Debug" to enter the debugger. Now press "Single Step" in the tool bar to single step through the code, observing how the values in the "Data", "Register" and "Memory" windows change as each step is executed. To execute a subroutine without entering it press "Step Over".

Suppose now we want to see how the code is executed when the key in the bottom right hand corner of the 4x4 matrix keypad is pressed (i.e. key number 00). Press "Reset Target" in the tool bar to begin execution at the first line of code, i.e. at "LDS #RAMEnd+1". Now scroll down the "Source" window to "RTIisr_2:", right click and select "Set Breakpoint". Then press "Start/Continue" to execute the code until "RTIisr_2:" is reached. Now press "Single Step" four times. The code that is about to be executed next is "LDAA PTP". Place your mouse arrow in the "Memory" window, right click and

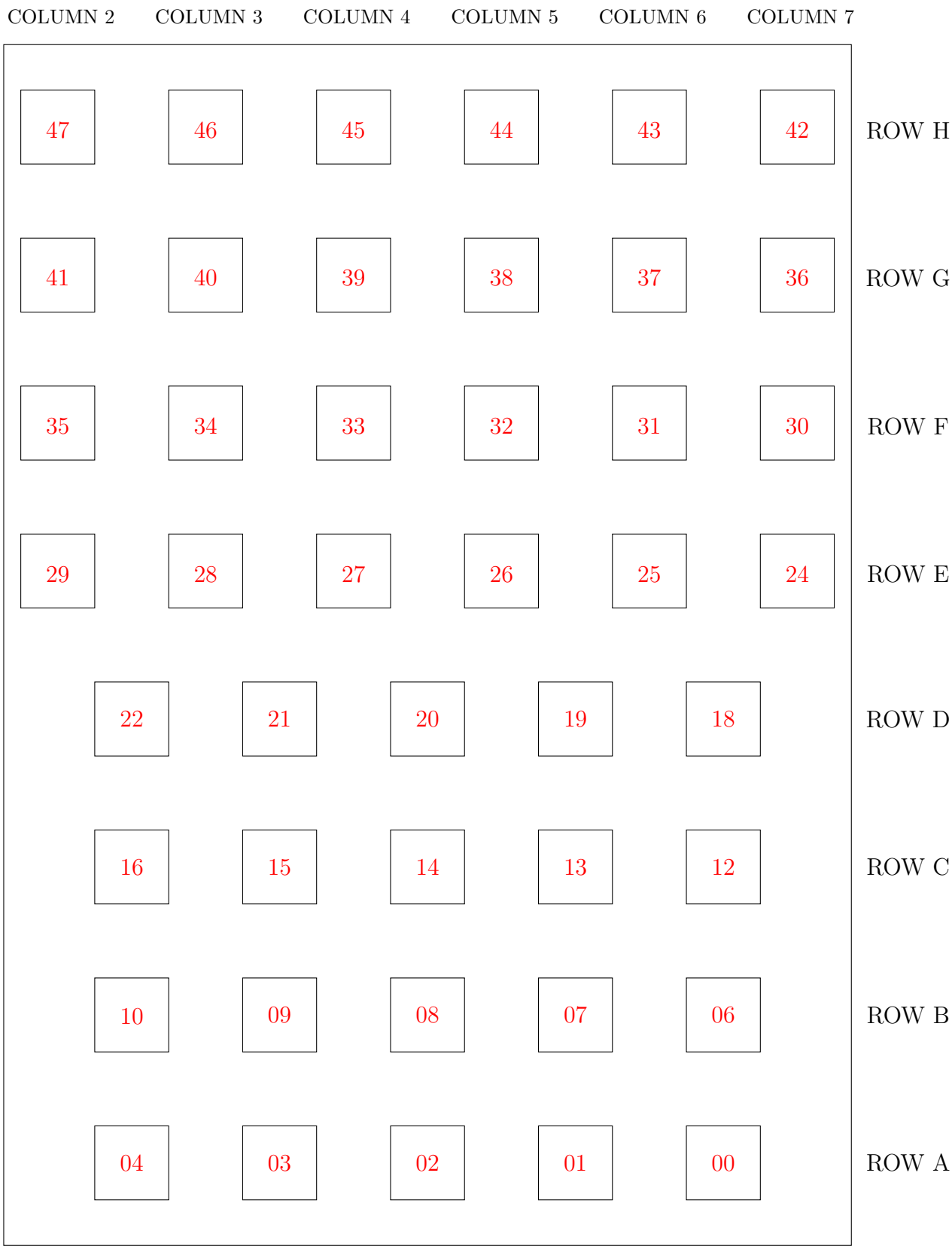

Figure 4.4: The 8x6 matrix keypad

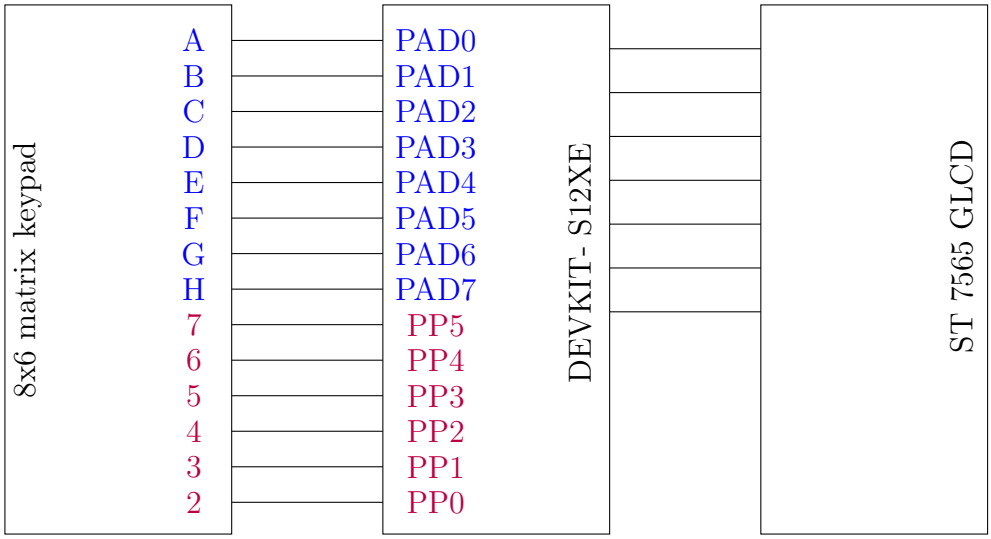

Figure 4.5: Interfacing the DEVKIT-S12XE to the 8x6 matrix keypad

select "Address". Enter "0258", the address of the Port P register (see page 1300 of the datasheet) and press "OK". Now key number 0 is in the first column, which is connected to pin 5 of Port P (PTP5). If this key is being pressed, then the voltage at PTP5 will be 0. Double click on the highlighted value at address 0258 in the "Memory" window and change it to "DF", then press "Enter" on your keyboard. Now press "Single Step" to load accumulator A with DF (the current value in accumulator A can be seen in the "Register" window). Press "Single Step" three more times to enter the "scanKey" subroutine. Scroll down to the first appearance of "JSR scanCol", right click and select "Run To Cursor", then press "Single Step" to enter the subroutine. The first instruction is "BRCLR PTP,#BIT5,scanCol_1", so pressing "Single Step" takes us to "scanCol_1:". Continue single stepping until you reach "key0:". Enter the subroutine "dispKeyNo" to see how it works, then use "Single Step" and "Step Over" to exit all subroutines and return to the interrupt service routine at "RTIisr_3:". Change the value of Port P in the "Memory" window to "FF" to indicate that key 00 has been released. Press "Start/Continue" to resume program execution. Another interrupt occurs and program execution halts at the breakpoint "RTIisr_2:". As an exercise, use the debugger to see how the code is executed when the key in the top left hand corner of the 4x4 matrix keypad is pressed (i.e. key number 15).

Note that KYPD1_FCS.asm contains a few code changes. Firstly Port P is set to output so that its value can be changed in the debugger to simulate pressing a key - this isn't possible if Port P is set to input. Secondly the SPI instructions in the subroutine "sendByte" are not executed in order to prevent an infinite loop, as no data is actually being sent.