Lab 4: Keypad, SRAM Expansion, and LSA

OBJECTIVES

• To understand how a keypad functions as a raster scan input device and to learn how to interface a keypad to a microprocessor.
• Further explore and understand the implementation of memory-mapped I/O (as you did in lab 3). Extend the processors memory capabilities by adding and external SRAM.
• To learn how to utilize the internal pull-up (or pull-down) resistors available on XMEGA I/O pins when used as inputs.
• You will also learn how to use a logic state analyzer (LSA). You will use your DAD as the LSA.

REQUIRED MATERIALS

• uPAD with Proto Base and tools
• Keypad
• Keypad Breakout Board (constructed in previous lab)
  o 1 – 8-pin male header for Breakout Board
  o 2 – 5-pin male headers for Breakout Board
• Keypad Connectors (constructed in previous lab)
  o 2 – 5-pin male headers for Proto Base
  o 2 – 8-pin male headers for Proto Base
• 32k x 8 ISSI SRAM chip
• XMEGA documents
  o doc8331 and doc0856
• Spec sheet for the 32k x 8 SRAM (ISSI)
• Digilent Analog Discovery (DAD) kit

YOU WILL NOT BE ALLOWED INTO YOUR LAB SECTION WITHOUT THE REQUIRED PRE-LAB.

PRELAB REQUIREMENTS

You must adhere to the Lab Rules and Policies document for every lab.

PART A – KEYPAD

To use the keypad, you will need 4 input pins and 4 output pins. You must use either port E or F for the keypad. Note that the chosen port must NOT have any other conflicting (input device) connected to it.

The keypad included in your kit is similar to a non-touchscreen cell phone keypad. See Figure 1 for a description of the keypad. Each key has two contacts, one attached to a “row” wire and the other attached to a “column” wire. When a key is pressed, the column wire and row wires are connected (with a small resistance). There is no power supplied to the keypad except through inputs to the row or column wires. Figure 2 shows how the keypad is interfaced with a microprocessor’s port(s). A pull-up or pull-down resistor is used on either each row wire or each column wire. Please note that if you use a keypad different from the one supplied in this semester’s kit, the pinouts may change. (The pinouts can be easily determined with a multimeter set on resistance.) You should verify the pin arrangement to be sure.

When the keypad is connected to the microprocessor and a key is pressed, a connection is established between an input port pin and an output port pin. A value written to the output port can be read from the input port. But what will be read on the unconnected (key not pressed) pins of the input port? Pull-up or pull-down resistors are used so that the input port pins do not have floating (i.e., indeterminate) values. The resistance is necessary to limit the current drawn by the port. With pull-up resistors, the input ports will be at “1” (Vcc = +3.3V) when no key is pressed. When a key is pressed, a “0” (GND) will appear on the input if the corresponding output is at “0”.

Unlike the external pull-up resistor used in Lab 2, you will use the XMEGA’s built in pull-up (or pull-down) resistors as mentioned in the I/O PORT section of the manual (doc8331). The relevant register on the input port is PINnCTRL (PORTx_PINnCTRL in the include file).
Lab 4: Keypad, SRAM Expansion, and LSA

Note that some of the pins on a port can be input and some can be used as output. The Multi-pin Port Configuration register, PORTCFG_MPCMASK, can be used to simplify your code for the keypad scan.

Note that in the below description, the words row and column can be reversed (with a corresponding modification to the circuit).

The keypad is read by scanning it one column at a time (Figure 2). First, write a 0b0111 (where 0b is a prefix representing binary numbers) to the appropriate output port pins (columns), and then read the input port (rows). This puts a zero on only the first column of the keypad. If any of the keys in column 1 are pressed, the corresponding input bits will be zero. If no key in column 1 is pressed, all inputs will be pulled high by the pull-up resistor. Next, write a 0b1011 to the output port pins (columns) to scan the second column. Continue by writing a 0b1101 then a 0b1110 to the output port pins to scan the third and fourth columns, respectively.

- **Warning:** Do not push two keys at once. Pressing two keys in the same row will connect output pins!

1. To implement the keypad interface, you need 4 input pins and 4 output pins. Normally, you could either use external I/O ports (like those from a previous lab) or a single 8-bit XMEGA port. But, because we want to use the internal XMEGA pull-up (or pull-down) resistors, you will have to use an XMEGA port for the inputs. You might as well use this same port for both the inputs and the outputs that the keypad requires.

2. Wire up your keypad as shown in Figure 2. Add a detailed schematic of keypad connections to the schematics done in previous labs and add this to your pre-lab report. (You will continue adding to this schematic throughout the semester.)

3. Write a subroutine to scan the keypad and determine the key pressed. Your subroutine should return one of 0x0 - 0xF, corresponding to the keypad keys. (Use 0xE for '*' and 0xF for '#'.) If no key is found, return 0xFF. **NOTE:** You will use your keypad extensively in future labs. Make your scan subroutine easily portable between programs.

4. Write a main routine (called Lab4_keypad.asm) to continuously scan the keypad (by calling the subroutine) and display the pressed key (0x0-0xF) on an LED bank. The LEDs should remain lit for as long as the key is pressed. If no key is pressed, display an LED pattern of your choice (such as an error code).

### Keypad Placement

You will connect the keypad to the uPAD Proto Base board using the supplied ribbon cable. The keypad breakout board will be soldered to the keypad board and the cable will connect to the keypad breakout board to the uPAD Proto Base board.

Cut two 8-pin and four 5-pin male headers from your header strips. You will insert one 8-pin and two 5-pin male headers through the **TOP** of the keypad breakout board (long side sticking up, perpendicular to the board), short side through the board. Solder one of the 5-pin headers as specified; make sure that this header is **as straight as possible**. Solder the second 5-pin header as specified; again make sure that this header is **as straight as possible**. Solder the 8-pin header.

Insert the 8-pin header up through the eight holes below the bottom row of keys on the keypad board, as shown in Figure 3. Solder these eight pins to secure the keypad break board to the keypad board.

Locate the J23 header socket on the uPAD Proto Base board. You will install two 5-pin headers into these holes. Insert the short pins of one 5-pin header through the board from the top and solder on the bottom; make sure that this header is **as straight as possible**. Insert the short pins of another 5-pin header through the board from the top and solder on the bottom; make sure that this header is **as straight as possible**.

Solder the remaining 8-pin header into J22. Insert the short end through the bottom of the board (so that you can wire-wrap on the bottom of the uPAD Proto Base). You will wire-wrap from the J22 header pins to the appropriate...
port pins.

[Note: The J21 and J20 header sockets are used for pull-up (or pull-down) resistors. Since we will use internal pull-up or pull-down resistors for the keypad, those will not be needed. If you were using external pull-up resistors, you would solder an 8-pin header into J21 as with J22, solder a SIP resistor into one of the columns of J20, and a 9-pin or 10-pin header into the other column of J20. You would also wire-wrap or connect jumpers across J21 to J22 for the 4 relevant pins requiring pull-up (or pull-down) resistors. Finally, you would wire-wrap from J20 header pins to the appropriate port pins.]

Figure 4 shows how you connect the keypad to the Proto Base using the ribbon cable in your kit.

Figure 4: Keypad connected to uPAD Proto Base

PART B – ADDING 32K OF SRAM
The purpose of this part of the lab is to interface an SRAM, a type of volatile memory, to your XMEGA. The internal SRAM of our XMEGA is only 8K, which is sufficient for every lab in this course if everything is done efficiently. However, some application may require additional SRAM. In the future labs, you are encouraged to store variables in the external SRAM.

The SRAM chip available in your lab kit is a 32k x 8 (32k bytes or 32K). We will be using the entire address space of this SRAM since we have multiple megabytes of external memory space available.

You will place 32k x 8 of SRAM in the system memory map beginning at \textbf{0x47 2000}. (Note that you cannot use spaces when you specify hex addresses in Atmel Studio, but it is easier to read when you do, so I use spaces when there are more than 4 digits in a hex address.) We may utilize addresses 0x47 0000 through 0x47 1FFF and 0x47 A000 through 0x47 FFFF in later labs.

5. Re-read through the EBI section in the XMEGA AU Manual (doc8331, section 27).
6. Configure CS1 (chip select 1) to enable \textbf{0x47 0000 – 0x47 FFFF} address range. Do not modify CS0 from the previous lab, i.e., your external I/O ports must still work.

7. Use your CPLD, CS1, and necessary address lines to place 32k x 8 SRAM in the data memory starting at \textbf{0x47 2000}. Your chip select must be configured as specified above.
8. Use Quartus to create a chip-enable equation for the SRAM. Add this chip-enable to your current working CPLD (bdf) schematic or VHDL. Don’t forget to submit your archive file (Lab4_RAM.qar) to Canvas with your other files.

Note: See Table 1 for the functioning of a typical SRAM. An SRAM can be disabled or tri-stated by not asserting the active-low CE. See the SRAM specification for more information.

Table 1: SRAM truth/voltage table.

<table>
<thead>
<tr>
<th>CE</th>
<th>WE</th>
<th>OE</th>
<th>Mode</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/H</td>
<td>-</td>
<td>-</td>
<td>High-Z</td>
<td>High-Z</td>
</tr>
<tr>
<td>1/L</td>
<td>0/H</td>
<td>0/H</td>
<td>High-Z</td>
<td>High-Z</td>
</tr>
<tr>
<td>1/L</td>
<td>0/H</td>
<td>1/L</td>
<td>Read</td>
<td>Data Out</td>
</tr>
<tr>
<td>1/L</td>
<td>1/L</td>
<td>-</td>
<td>Write</td>
<td>Data In</td>
</tr>
</tbody>
</table>

9. Program your CPLD with the modified (bdf) schematic (or VHDL file) and verify that your board still boots. If the board does not boot, verify that your CPLD SRAM equation is correct and that your file contains the equations for the other components.
10. Using the datasheet for the 32k x 8 SRAM, create a schematic showing all the necessary connections to the processor board. Do not draw wires. Use pin labels instead. Include all pin numbers. Be sure to use the pinout labelled “28-pin SOP”, not TSOP!
11. There should only be one remaining surface mount island (the far right island in Figure 5), so place it there. Hold the SRAM chip in place on the board by soldering two corners first. After soldering the SRAM in place, solder a 0.1\mu F bypass capacitor in the respective bypass capacitor location. Finally, solder on the headers so that you may wire-wrap to the SRAM.

Figure 5: Locations (from left to right) of the 74’574, 74’573, and 32k x 8 SRAM on uPAD Proto Base.
Lab 4: Keypad, SRAM Expansion, and LSA

12. It is important to note that before you started this lab you already have two sets of data bus pins wire wrapped to the Proto Base data bus header. Instead of wire wrapping a third set from this header directly to the SRAM’s data pins, you should use one of the daisy chaining method as shown in Figure 6 and Figure 7. To daisy chain, connect pins to the same node, but not necessarily the same header pins.

![Figure 6: A daisy chain solution.](image)

![Figure 7: A better daisy chain solution.](image)

13. Wire the SRAM to the address, data, RE, WE, and CE signals. Use a multimeter to verify your connections as you wire them. Make sure that no signals are mistakenly tied to power or ground. It is easy to make a mistake wire-wrapping the SRAM. Proceed slowly and carefully. After the SRAM is wired, power on the board and verify that you can still program the processor.

14. If you have not already noticed, the index registers X, Y, and Z are all 16-bits while the address of the external SRAM is 24-bits (e.g., 0x47 2000). The processor has additional registers to hold the value of this upper byte that we are missing: CPU_RAMPX, CPU_RAMPY, and CPU_RAMPZ. You can use an OUT instruction to load these registers (instead of STS). See the Atmel Instruction Set (doc0856) and XMEGA AU Manual (doc8331, section 3.10, 3.14, and 3.15) for more information.

15. Write an program (called Lab4_SRAM.asm) to test the SRAM:
   a. Write 0xAA to all 32k addresses starting at 0x47 2000.
   b. Re-read all addresses and check that 0xAA was written. If not, write the lower 16 bits (ignore RAMP) of all of the corrupt addresses to internal SRAM starting at 0x2200. Store the corrupt addresses in little-endian format (i.e., the low byte followed by the high byte). After 100 errors, you can stop recording.
   c. Write 0x55 to all 32k addresses.
   d. Repeat step b, except this time check for 0x55 and store the first 100 errors at 0x2400.
   e. End your program with an endless loop.
   f. Your program should store only the lower 16 bits of corrupt addresses. Make your program modular so that it is easy to put breakpoints between steps.

16. By default, it is not possible to view external memory in Atmel Studio. Usually, it can be made possible with the following simple procedure. Do this procedure before coming to lab; it will make your debugging/testing easier and also help you TA if something is not working correctly. The instructions are available on our website at [mil.ufl.edu/3744/docs/External_Memory_Viewing.pdf](mil.ufl.edu/3744/docs/External_Memory_Viewing.pdf).

PRE-LAB QUESTIONS
1. What is the size of CS1 used in this lab?
2. Which address lines MUST be present in the SRAM chip-enable equation for full address decoding?
3. Draw a schematic of an 8k x 8 RAM expansion starting at address 0x9000. You may only use address lines for decoding.
4. On the uPAD Proto Base board, is the CPLD required for interfacing with the SRAM? If not, explain.
5. The simple memory test program described above is not very good. It checks for neighboring data pins that are shorted or left unconnected, but we have no idea if the address bus is working. Describe a procedure for testing the address lines. Are there any limitations to your procedure (for example, does your procedure test all of the address lines)? Explain.
6. In this lab, we configure CS1 to be bigger than the SRAM so that we can divide it for multiple devices. If we want to connect many devices to the memory bus, this is a good way to conserve XMEGA chip select outputs. Suppose we wanted to configure CS1 to span from 0x46 0000 to 0x47 FFFF instead. Is this a valid range for an XMEGA chip select? With the hardware on your uPAD Proto Base as it is now, can we divide this new CS1 the same way we do in this lab? Why or why not?

PRE-LAB REQUIREMENTS
1. If necessary, add the keypad circuits to your uPAD Proto Base board.
2. Add the 32k x 8 SRAM to your board.
3. Write the required programs.

IN-LAB REQUIREMENTS
1. Demonstrate that your keypad works properly.
2. Demonstrate that your SRAM works properly. Put break points between steps 15a, 15b, 15c and 15d. Your TA will corrupt several memory locations to make sure your program works.
3. Utilize your DAD (Digilent Analog Discovery) board as an LSA (Logic State Analyzer). Demonstrate the use of the logic analyzer feature of your DAD by
probing some of the EBI pins, address pins, data pins, etc. I suggest you try this at home (i.e., prior to coming to lab).

4. Answer any questions your TA may have about your lab.

DEBUGGING TIPS:

1. Your DAD and your multimeter can be helpful when debugging the keypad. You can set breakpoints in your scanning routine and check the outputs and inputs using the probes.

2. If you are using the same pointer (X, Y, or Z) to read from two different memory-mapped devices, make sure to re-initialize or clear the corresponding RAMP register.

3. If you are using external I/O ports for the keypad, you may need to include up to 2 NOPs between reading and writing from/to the same address. This will flush the instruction pipeline and ensure that these instructions function correctly, i.e., as expected. Do not add the NOPs unless they are needed. If they are needed, please let Dr. Schwartz and your TA know and then them your code.

4. It is recommended (and may be possible) to configure your Atmel Studio to display the SRAM contents in the memory browser of the debugger. See mil.ufl.edu/3744/docs/External_Memory_Viewing.pdf for instructions on how to try this. If you try this, please let Dr. Schwartz and your TA know if it works or not.