

GH09.B.1.portkey - Port Expander and Keypad Controller

General Description

This benchmark is a general purpose input output (GPIO) port expander and keypad controller. This is a slave device that communicates to a main controller via a serial peripheral interface (SPI) bus. There are 16 GPIO pins that can either be programmed as inputs, outputs, or part of a keypad controller that can be configured to a maximum of an 8x7 keypad matrix. Each GPIO can be programmed to send a general interrupt signal to the master device via a common interrupt output at which the device needs to be queried as to which GPIO caused the interrupt.

Features

- SPI 4 pin bus to master
- 16 programmable pins
- 8x7 keypad controller
- Programmable interrupts when in GPIO mode

Block Diagram

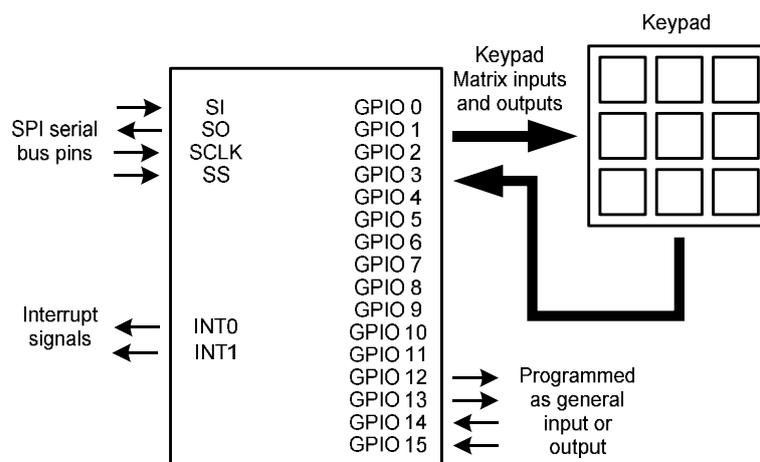


Figure 1 –Sample configuration of the port expander

Details

The port expander and keypad is designed to expand the number of I/Os from a master device that has a limited number of pins for input and output.

Design Specs [Optional design choices]

Registers

The suggested register file for this device is 5 registers 16bits wide. Registers 0x00 through 0x04 are described below.

Figure 2 shows some of the registers the in the device that control the setup. Register (a), called the *Keypad Control Register*, uses the Keypad bit (bit 0) to determine if the keypad control registers are in use or not. Row0 to Row3 and Col0 to Col2 are loaded with the size of the keypad matrix. For example, 0xF2 means that the keypad is in use and has 7 output columns and 2 input rows that are in use for a 7x2 keypad matrix. The first 9 pins, GPIO 0 – GPIO 8, will be used for this keypad matrix, and the first 7 pins will be programmed as outputs and the last 2 pins as inputs. The remaining 7 GPIO pins can be used as general I/O pins noting that any general purpose register functions that try to affect the first 9 pins will be masked out. For example, trying to make pin 0 an interrupt would have no effect. The 8 unused bits (bit 8-15) of the *Keypad Control Register* are not used for anything.

Register (b) and (c) in Figure 2 are the *Interrupt Register*. For any GPIO pins in input mode they can be programmed to send an interrupt if the input value on the pin changes. Note that if a GPIO pin is being used as part of the keypad matrix that the values in these registers have no effect.

Registers (d) and (e) in Figure 2 are the *Direction registers*. This determines the direction of the GPIO pin as either an input pin or an output pin. Note that if a GPIO pin is being used as part of the keypad matrix that the values in these registers have no effect, since the keypad configuration overrides the direction of these registers.

Register (f) and (g) hold the values of the GPIO pins. If a read is performed on these registers the master must know which pins are in input mode, since the read of GPIOs programmed as output ports has no meaning. Similarly, a write to these registers will not overwrite GPIO values that are set to read (this is likely determined by masking a write with the Direction register).

In addition to the registers shown, there is a need for a FIFO that contains a record of the key-presses detected on the keypad matrix. This FIFO buffer can buffer 16 key-presses. A read on address 0x04 will cause the device to send a 16 bit result (on the SI output) for the keypad

coordinates of the front of the FIFO of keypresses. The first 8 bits describe the row pressed and the second 8 bits describe the column pressed.

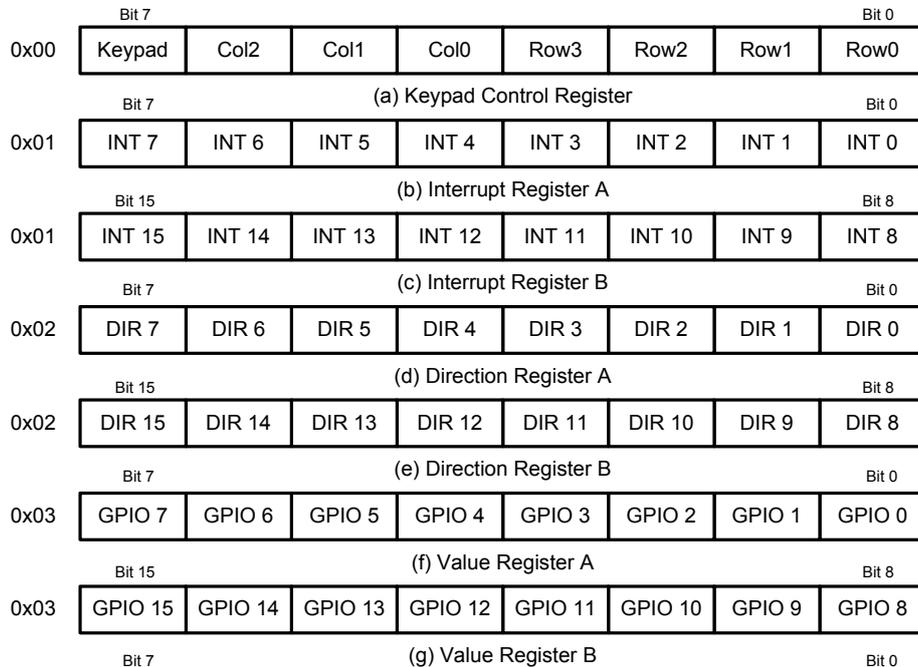


Figure 2 – The potential control registers

Serial Bus Communication

The Serial Peripheral Interface (SPI) interface has the following specifications for this chip acting as a slave on the SPI.

Master write

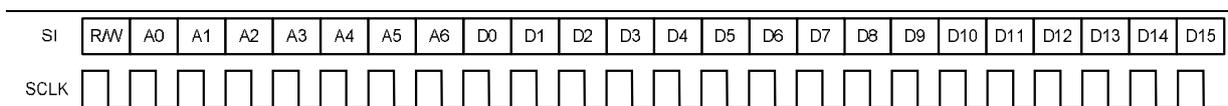


Figure 3 – A write message from the master

The master can write to any of the registers with a packet similar to the one in Figure 3. The first bit is set to indicate a write message. Address bits A6-A0 tell the slave which register is to be written to. D15-D0 contains the bits that are to be written.

The pin SS is set to tell this device that the packet on the bus is meant for it. This is not shown in the timing diagram in Figure 3.

Read

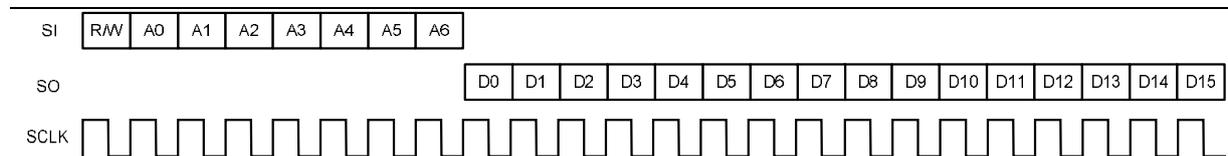


Figure 4 – A read message from the master and reply from the slave

The read on the serial bus is shown in Figure 4. Again, the master sends a bit to indicate a read and the address of the register to be read in bits A6-A0. Once the 8 bits are sent by the master, the slave responds on the clock with the 16 bits (D15-D0) of the requested register.

The slave knows that this read request is for it based on the setting of the SS signal.

Pins

There are two interrupt pins coming from the device. One interrupt pin is dedicated to the keypresses, and the second interrupt is dedicated to GPIO interrupt pins. The keypress interrupt details are described below. An interrupt caused by a GPIO interrupt means that a value has changed on an input pin that is set to interrupt (programmed by the *Interrupt Register*). It is the responsibility of the master to read the value of all the read registers and determine which read input pins have changed in value.

Keypad Controller

The rows in the keypad are set as inputs and the columns of the keypad pins are output pins. The basic operation of the keypad controller is to have either a high signal going out or a low signal going out (in the case of a low output the input registers will need to be tied to a high voltage) on all the output pins. Then when a key is pressed on the matrix, the switch connects the circuit and a change is read on the input pins. This indicates that at least one key is being pressed.

The controller proceeds to test each output matrix line individually by uniquely setting each of these outputs and checking to see which inputs are active. In this way the column and row coordinates tell you which key is being pressed (or possibly ghosting).

Each key press coordinates is recorded in the fifo buffer, and an interrupt signal is sent to indicate that the keypad was pressed. Each interrupt should be accompanied by a read by the master to take the keypress off the buffer. If the 16 keypress buffer is full, then no additional keypresses will be added to the buffer, and will simply be lost.

Assumption: Max keypress speed is ~20 keypresses/second