PIC32 I/O

E155

Outline

- GPIO
- SPI
- UART

Ref. PIC Family Reference Manual: http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en545644

GPIO

- PIC32 organizes groups of GPIOs into ports that are read and written together.
- Our PIC32 calls these ports RA, RB, RC, RD, RE, RF, and RG.
- A port may have up to 16 GPIO pins, although the PIC32 doesn't have enough pins to provide that many signals for so many ports.
- The number of GPIO pins available depends on the size of the package.

GPIO Pins

- Table 8.1 summarizes which pins are available in various packages. For example, a 100-pin TQFP provides pins RA[15:14], RA[10:9], and RA[7:0] of port A. Note that RG[3:2] are input only. Also, RB[15:0] are shared as analog input pins, and most of the other pins have multiple functions as well.

Table: PIC32MX5xx/6xx/7xx GPIO pins

Port	64-pin QFN/TQFP	100-pin TQFP, 121-pin XBGA
RA	None	15:14, 10:9, 7:0
RB	15:0	15:0
RC	15:12	15:12, 4:0
RD	11:0	15:0
RE	7:0	9:0
RF	5:3, 1:0	13:12, 8, 5:0
RG	9:6, 3:2	15:12, 9:6, 3:0

GPIO Pins

- The logic levels are LVCMOS-compatible. Input pins expect logic levels of $V_{IL} = 0.15V_{DD}$ and $V_{IH} = 0.8V_{DD}$, or 0.5 and 2.6 V assuming V_{DD} of 3.3V.
- Output pins produce V_{OL} of 0.4 and V_{OH} of 2.4 V as long as the output current does not exceed 7 mA.

Special Function Registers

- TRIS
  - Data Direction or Tri-State Control register that determines whether a digital pin is an input or an output.
    - TRISx register bit = 1, configures the corresponding I/O pin as an input

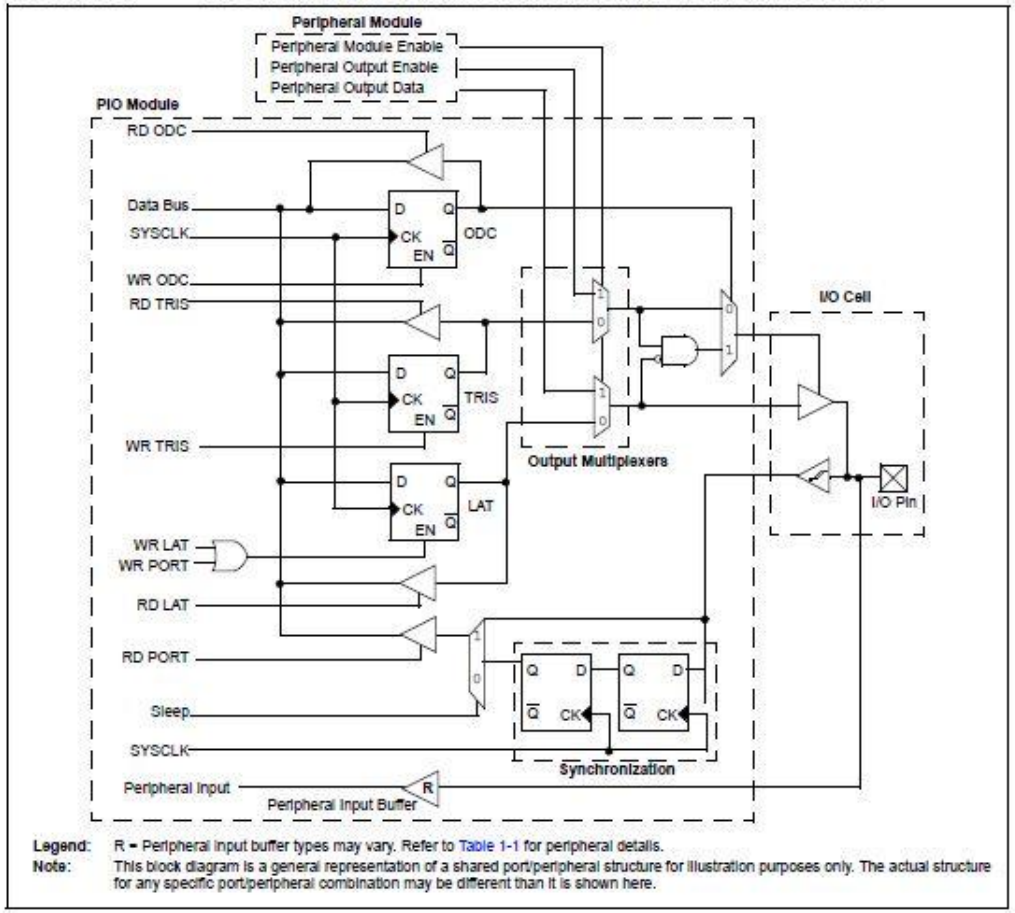
      where x is a letter (A-G) indicating the port of interest.
    - TRISx register bit = 0, configures the corresponding I/O pin as an output.

Special Function Registers

- PORT is a register used to read the current state of the signal applied to the port I/O pins.
- Writing to a PORTx register performs a write to the port's latch, LATx register, latching the data to the port's I/O pins.
- LAT is a register used to write data to the port I/O pins. The LATx Latch register holds the data written to either the LATx or PORTx registers. Reading the LATx Latch register reads the last value written to the corresponding PORT or Latch register.

GPIO

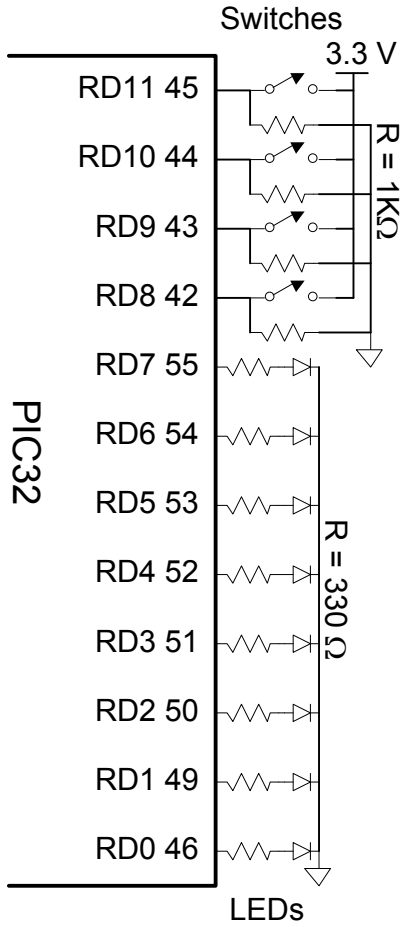
FIGURE 12-1: BLOCK DIAGRAM OF A TYPICAL MULTIPLEXED PORT STRUCTURE



Legend: R = Peripheral input buffer types may vary. Refer to Table 1-1 for peripheral details.
Note: This block diagram is a general representation of a shared port/peripheral structure for illustration purposes only. The actual structure for any specific port/peripheral combination may be different than it is shown here.

Microchip Technology: http://ww1.microchip.com/downloads/en/DeviceDoc/61156G.pdf

GPIO Example



- The LEDs are wired to glow when driven with a 1 and turn off when driven with a 0.
- The switches are wired to produce a 1 when closed and a 0 when open.
- The microcontroller can use the port to drive the LEDs and read the state of the switches.

Read SW Write LED

```
#Configure TRISD so that pins RD[7:0] are outputs and RD[11:8] are
inputs.
#Read the switches by examining pins RD[11:8]
#Write this value back to RD[3:0] to turn on the appropriate LEDs.
#Assembly code:
   la      $t0, TRSD       # load address of TRSD ->$t0
   addi    $t1, $0, OxFF00
   sw      $t1, 0($t0)     # TRISD=F00, configure RD[7:0] as outputs and
RD[11:8] as inputs
   la      $t0, PORTD      # load address of PORTD
   lw      $t1, 0($t0)     # t1=PORTD, read SW1-SW4 (RD[11:8])
   srl     $t1, $t1, 8     # >>8
   andi    $t1, $t1, 0x000F #mask off RD[7:4]
   sw      $t1, 0($t0)     # write to LED[7:0]
```

Serial I/O

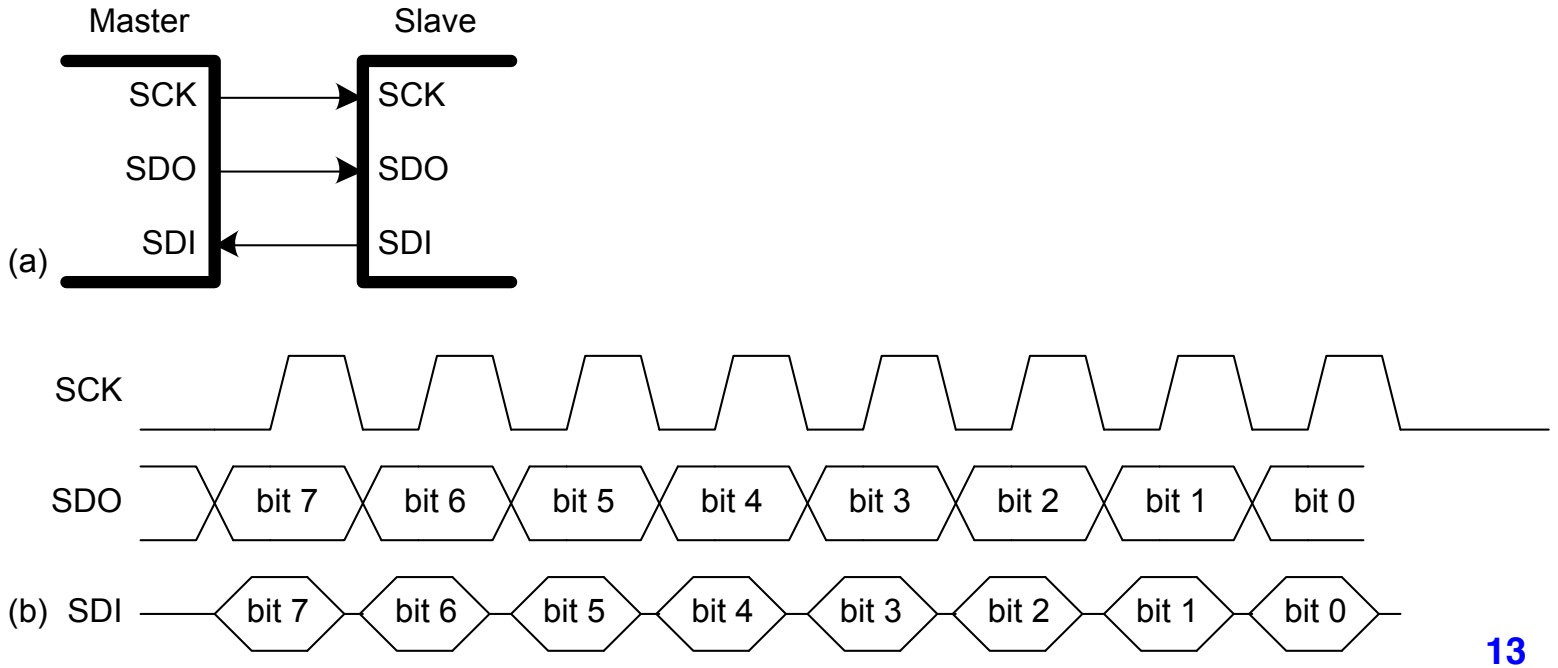
- Data inputs and outputs are in general broken into pieces that are input or output one at a time.
- *Serial* I/O is 1 bit at a time and *parallel* I/O is several bits (e.g. 16 bits).
- Serial I/O is popular because it uses few wires and is fast enough for many applications.
- Multiple standards for serial I/O have been established and the PIC32 has dedicated hardware to easily send data via these standards.
  - SPI: Serial Peripheral Interface
  - UART: Universal Asynchronous Receiver/Transmitter
  - I^2C: Inter-Intergrated Circuit
  - USB: Universal Serial Bus
  - Ethernet

SPI

- SPI is a simple synchronous serial protocol that is easy to use and relatively fast.
- The physical interface consists of three pins:
  - Serial Clock (SCK)
  - Serial Data Out (SDO)
  - Serial Data In (SDI).
- SPI connects a *master* device to a *slave* device. The master produces the clock. It initiates communication by sending a series of clock pulses on SCK. If it wants to send data to the slave, it puts the data on SDO, starting with the most significant bit.

SPI

- SPI connects a *master* device to a *slave* device, as shown.
- The master produces the clock. It initiates communication by sending a series of clock pulses on SCK. If it wants to send data to the slave, it puts the data on SDO, starting with the most significant bit.



SPI

SPI CON register fields

- Each SPI port is associated with four 32-bit registers: SPIxCON, SPIxSTAT, SPIxBRG and SPIxBUF.
- SPI1CON is the control register for SPI port 1. It is used to turn the SPI ON and set attributes such as the number of bits to transfer and the polarity of the clock.
- Slide 15 lists the names and functions of all the bits of the CON registers.
- STAT is the status register indicting, for example, whether the receive register is full.
- The detailed of registers are described in the PIC data sheet.

SP1CON Register

Bits	Name	Function
31	FRMEN	1: Enable framing
30	FRMSYNC	Frame sync pulse direction control
29	FRMPOL	Frame sync polarity (1 = active high)
28	MSSEN	1: Enable slave select generation in master mode
27	FRMSYPW	Frame sync pulse width bit (1 = 1 word wide, 0 = 1 clock wide)
26:24	FRMCNT[2:0]	Frame sync pulse counter (frequency of sync pulses)
23	MCLKSEL	Master clock select (1 = master clock, 0 = peripheral clock)
22:18	unused	
17	SPIFE	Frame sync pulse edge select
16	ENHBUF	1: Enable enhanced buffering
15	ON	1: SPI ON
14	unused	
13	SIDL	1: Stop SPI when CPU is in idle mode
12	DISSDO	1: disable SDO pin
11	MODE32	1: 32-bit transfers
10	MODE16	1: 16-bit transfers
9	SMP	Sample phase (see Figure 8.10)
8	CKE	Clock edge (see Figure 8.10)
7	SSEN	1: Enable slave select
6	CKP	Clock polarity (see Figure 8.10)
5	MSTEN	1: Enable master mode
4	DISSDI	1: disable SDI pin
3:2	STXISEL[1:0]	Transmit buffer interrupt mode
1:0	SRXISEL[1:0]	Receive buffer interrupt mode

Microchip Technology: http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en545644

SPI Data Rate

- BRG is the baud rate register that sets the speed of SCK relative to the peripheral clock according to the formula:

$$f_{SPI} = \frac{f_{peripheral-clock}}{2 \times (\mathrm{BRG}+1)}$$

- BUF is the data buffer. Data written to BUF is transferred over the SPI port on SDO pin, and the received data on SPI pin can be found by reading BUF after the transfer is complete.

SPI Operation

1. To prepare the SPI in master mode, first turn it OFF by clearing bit 15 of the CON register (the ON bit) to 0.
2. Clear anything that might be in the receive buffer by reading the BUF register.
3. Set the desired BAUD rate by writing the BRG register. For example, if the peripheral clock is 20 MHz and the desired BAUD rate is 1.25 MHz, set BRG to 20/(2×1.25) – 1 = 7.
4. Put the SPI in master mode by setting bit 5 of the CON register (MSTEN) to 1.
5. Set the bit 8 of the CON register (CKE) so that SDO is centered on the rising edge of the clock.
6. Finally, turn the SPI back ON by setting the ON bit of the CON register.

SPI Operation

- To send data to slave:
  - Write the data to BUF register
- Data will be transmitted serially
- Slave simultaneously send data back to the master
- Wait until bit 11 of the STAT register (the SPIBUSY bit) becomes 0 indicating that the data received from slave can be read from BUF

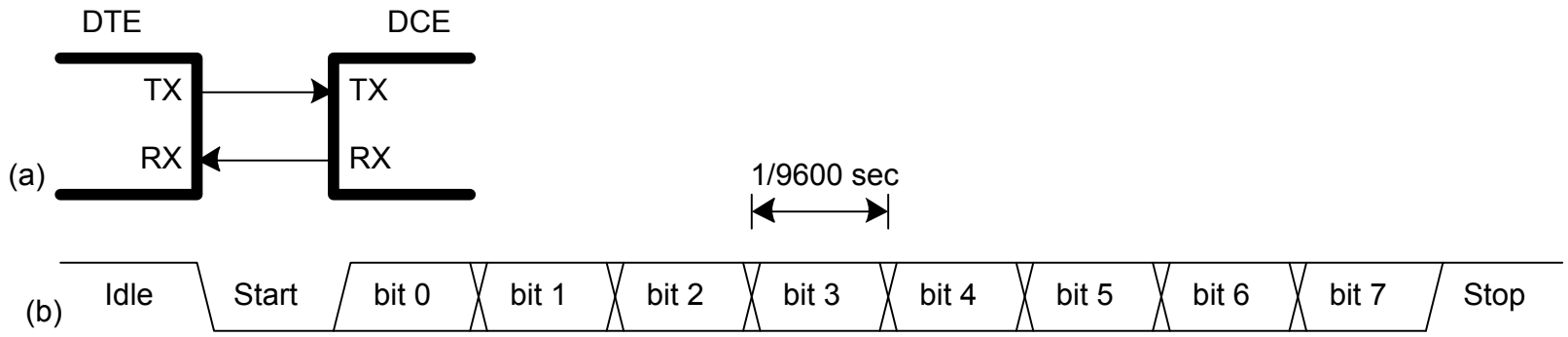
UART

- A UART is a serial I/O peripheral that communicates between two systems without sending a clock.
- Instead, the systems must agree in advance about what data rate to use and must each locally generate its own clock.
- Although these system clocks may have a small frequency error and an unknown phase relationship, the UART manages reliable asynchronous communication.

UART

- UARTs are used in protocols such as RS-232 and RS-485. For example, computer serial ports use the RS-232C standard, introduced in 1969 by the Electronic Industries Association.
- The standard originally envisioned connecting *Data Terminal Equipment* (DTE) such as a mainframe computer to *Data Communication Equipment* (DCE) such as a modem.
- Although UARTs relatively slow compared to SPI, the standards have been around for so long that they remain important today.

UART Operation



- Line idles at a logic '1' when not in use
- Each character is sent as a start bit (0), 7-8 data bits, and optional parity bit, and one or more stop bit (1)
- UART detects the falling transition from idle to start to lock on to the transmission to synchronize data transmission

UART Operation

- The *parity* bit allows the system to detect if a bit was corrupted during transmission. It can be configured as *even* or *odd*.
- A common choice is 8 data bits, no parity, and 1 stop bit, making a total of 10 bits to convey an 8-bit character of information.
- Data rates are referred to in units of baud rather than bits/sec.
  - 9600 baud indicates 9600 bit times / sec
  - or 960 characters / sec
  - giving a data rate of 960 × 8 = 7680 data bits / sec.

UART Operation

- Typical baud rates include 300, 1200, 2400, 9600, 14400, 19200, 38400, 57600, and 115200.
- The lower rates were used in the 1970's and 1980's for modems that sent data over the phone lines as a series of tones.
- In contemporary systems, 9600 and 115200 are two of the most common baud rates; 9600 is encountered where speed doesn't matter, and 115200 is the fastest standard rate, though still slow compared to other modern serial I/O standards.

UART

- The PIC32 has six UARTs named U1-U6.
- Each UART is associated with five 32-bit registers:
  - UxMODE: configure

    Default to 8 data bits, 1 stop, no RTS/CTS flow control

    bit15 is ON control to enable UART
  - UxSTA: status

    Set the UTXEN and URXEN bits (bits 10 and 12) to enable TX and RX pins

    UTXBF (bit 9) indicates that the transmit buffer is full

    URXDA (bit 0) indicates that the receive buffer has data available
  - UxBRG: set Baud rate
  - UxTXREG: data transmission start when written
  - UxRXREG: data receiving start when read

UART

– UxBRG: set Baud rate

set the baud rate to a fraction of the peripheral bus clock

$$f_{UART} = \frac{f_{peripheral-clock}}{16 \times (\mathrm{BRG}+1)}$$

BRG Settings for a 20 MHz peripheral clock

Target Baud Rate	BRG	Actual Baud Rate	Error
300	4166	300	0.0%
1200	1041	1200	0.0%
2400	520	2399	0.0%
9600	129	9615	0.2%
19200	64	19231	0.2%
38400	32	37879	-1.4%
57600	21	56818	-1.4%
115200	10	113636	-1.4%

UART

- UxTXREG: data transmission start when written

  To transmit data, wait until STA.UTXBF is clear indicating that the transmit buffer has space available, and then write the byte to TXREG

- UxRXREG: data receiving start when read

  To receive data, check STA.URXDA to see if data has arrived, and then read the byte from the RXREG.