

DEEPAK GUPTA



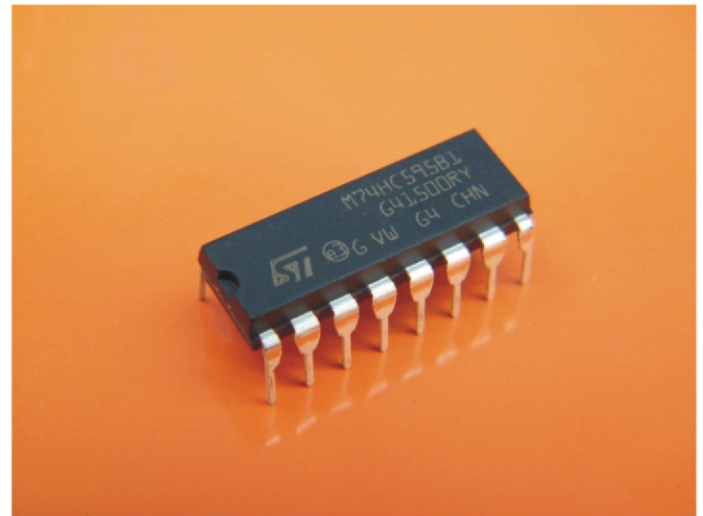
PORT EXPANDER

74HC595

Frontline Electronics,
Pvt. Ltd., Salem

Sometimes we need to extend or add more I/O ports to our microcontroller based project. Because usually we only have a limited I/O port left than the logical choice is to use the serial data transfer method; which usually only requires from one up to four ports for doing the data transfer. Currently there are few types of modern embedded system serial data transfer interface widely supported by most of the chip's manufactures such as I2C (read as I square C), SPI (Serial Peripheral Interface), 1-Wire (One Wire), Controller Area Network (CAN), USB (Universal Serial Bus) and the RS-232 families (RS-423, RS-422 and RS-485). The last three interface types is used for long connection between the microcontroller and the devices, up to 1200 meters for the RS-485 specification, while the first three is used for short range connection.

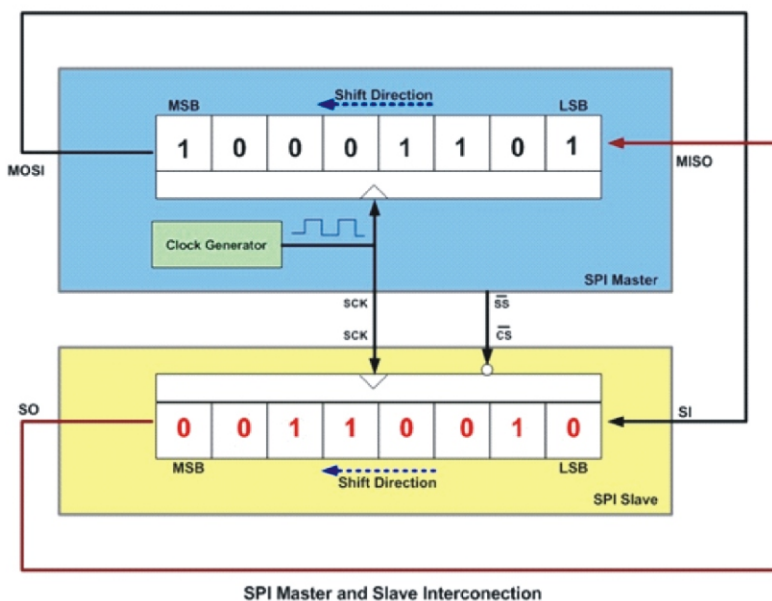
Among these serial data transfer interface types, SPI is considered the fastest synchronous with full duplex serial data transfer interface and can be clocked up to 10 MHz; that is why it is widely used as the interface method to the high speed demand



peripheral such as the Microchip Ethernet controller ENC28J60, Multi Media Card (MMC) Flash Memory, Microchip SPI I/O MCP23S17, Microchip 128K SPI EEPROM 25AA128, ADC, sensors, etc. The 74HC595 device has 8-bit serial-in, parallel-

out shift register that feeds directly to the 8-bit D-type storage register. It makes use of SPI protocol.

SPI stands for Serial Peripheral Interface. SPI is frequently used when few I/O lines are available, but communication between two or more devices must be fast and easy to implement. SPI is a synchronous protocol that allows a master device to initiate communication with a slave device. Data is exchanged between these devices. The clock signal is provided by the master to provide synchronization. The clock



signal controls when data can change and when it is valid for reading. Since SPI is synchronous, it has a clock pulse along with the data. RS-232 and other asynchronous protocols do not use a clock pulse, but the data must be timed very accurately. Since SPI has a clock signal, the clock can vary without disrupting the data. The data rate will simply change along with the changes in the clock rate. This makes SPI ideal when the microcontroller is being clocked imprecisely, such as by a RC oscillator.

Circuit Diagram

The circuit shows the LPC2120 connection with a shift register, 74HC595 which is further connected to a seven segment module. The interfacing used is SPI. The data on the seven segment is displayed using the SPI protocol. The circuit is built on proteus and crystal frequency has been internally set and also sends a string on virtual terminal. As shown,

using three pins for 74HC595, we are able to get eight port pins. Using multiple 74HC595, we can further expand port pins.

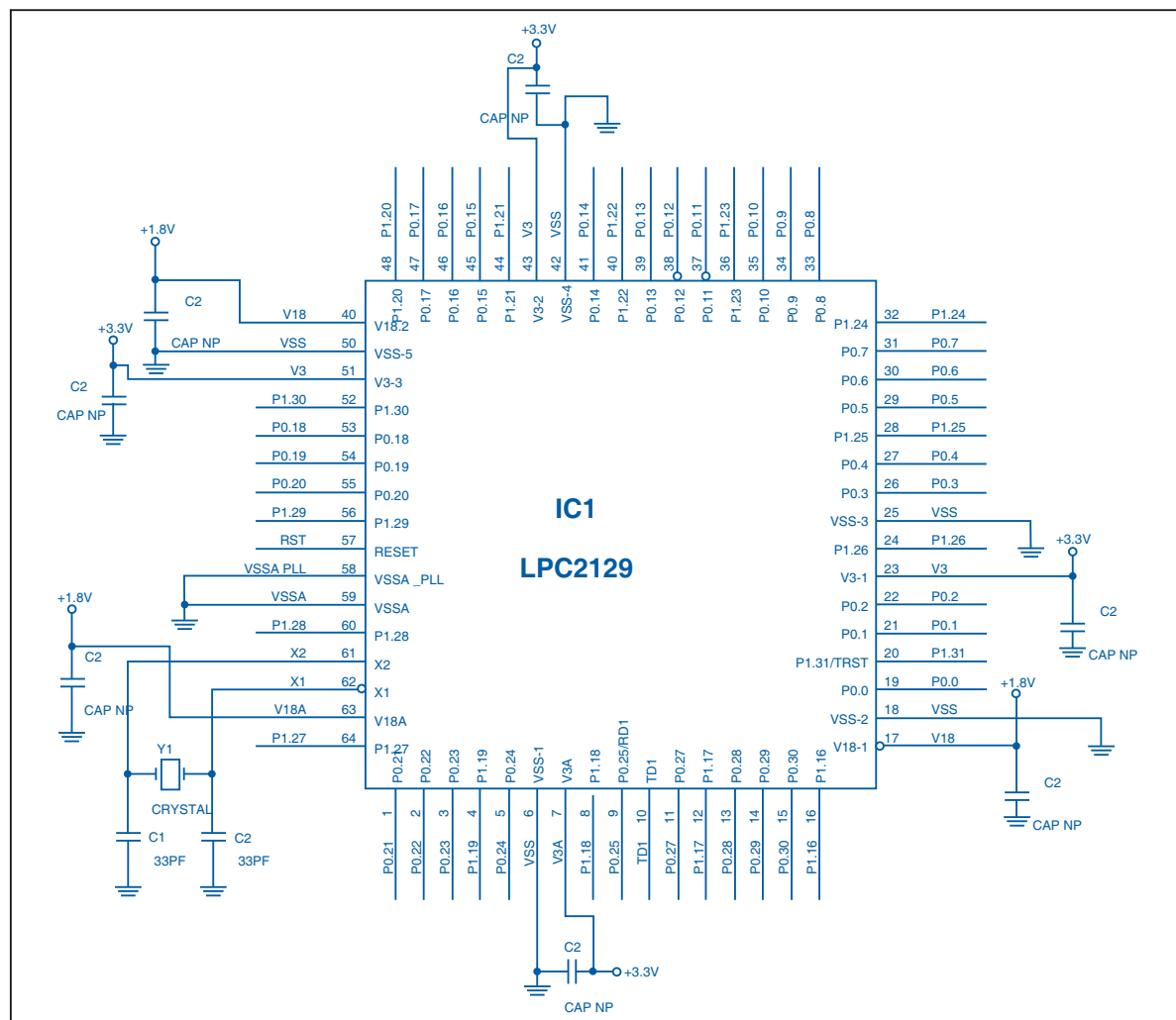
Component Used

LPC2129

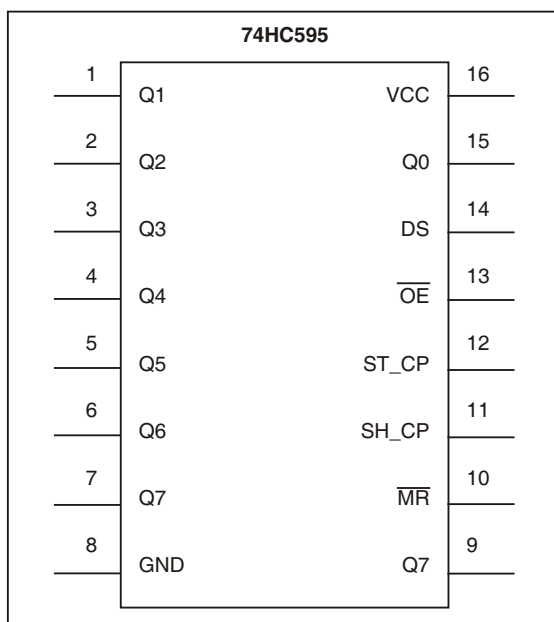
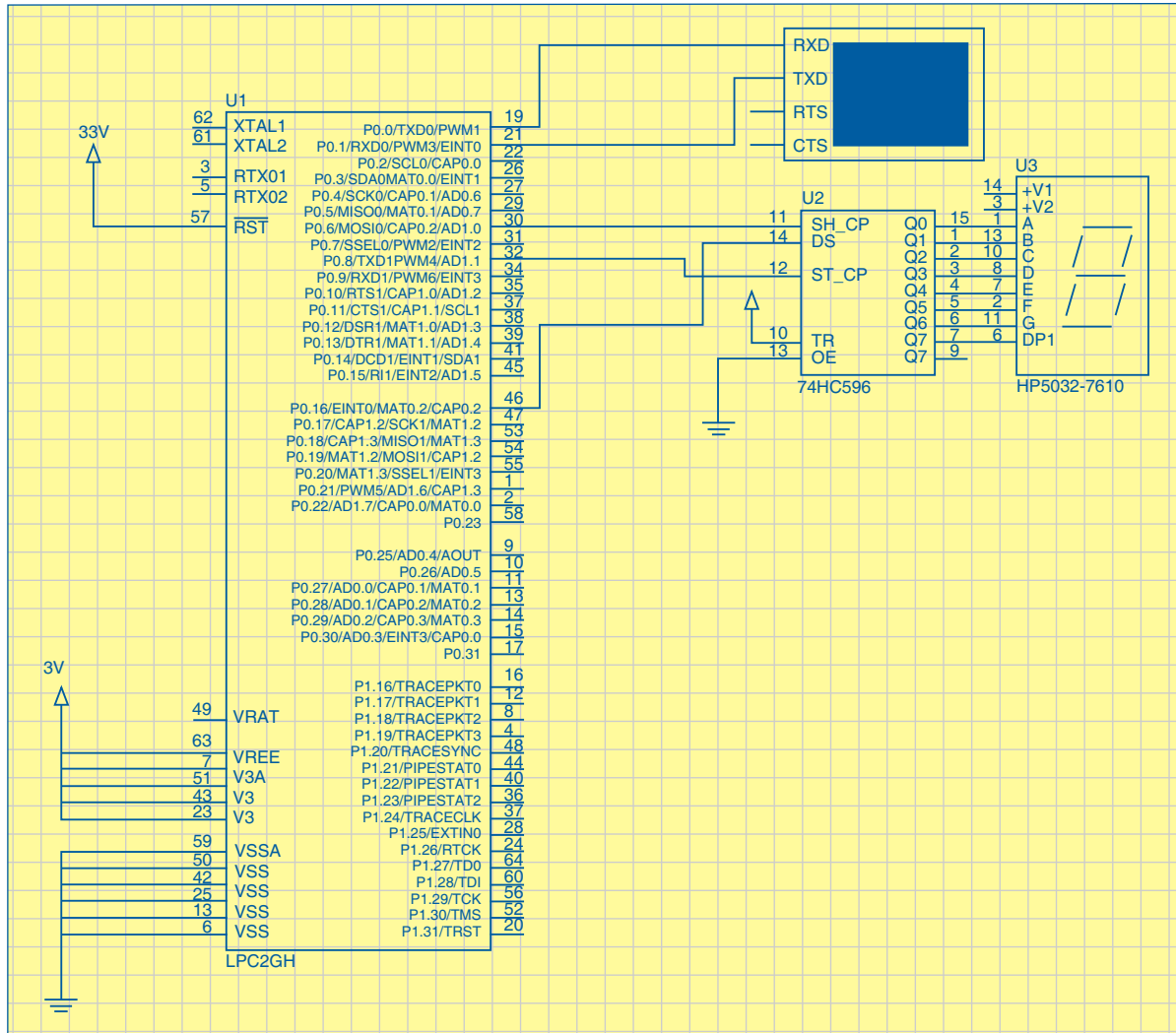
The LPC2129 are based on a 16/32 bit ARM7TDMI-S™ CPU with real-time emulation and embedded trace support, together with 128/256 kilobytes (kB) of embedded high speed flash memory.

Some Key features

- 16/32-bit ARM7TDMI-S microcontroller in a tiny LQFP64 package.
- 16 kB on-chip Static RAM.
- 128/256 kB on-chip Flash Program Memory.
- 128-bit wide interface/accelerator enables high speed 60 MHz operation.
- Two interconnected CAN interfaces with advanced acceptance filters.



CONSTRUCTION



- Four channel 10-bit A/D converter with conversion time as low as 2.44 ms.
- Multiple serial interfaces including two UARTs (16C550), Fast I2C (400 kbits/s) and two SPIs
- 60 MHz maximum CPU clock available from programmable on-chip Phase-Locked Loop with settling time of 100 ms.
- Vectored Interrupt Controller with configurable priorities and vector addresses.
- Two 32-bit timers (with four capture and four compare channels), PWM unit (six outputs), Real Time Clock and Watchdog.

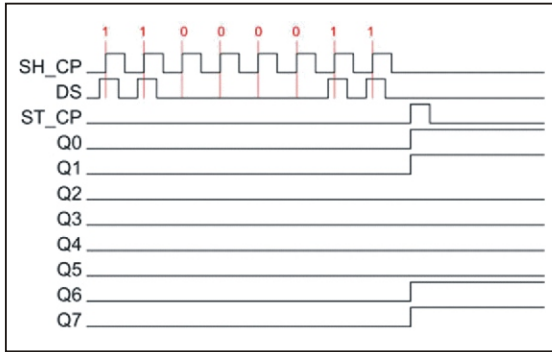
74HC595

Pins Description

Q0-Q7 8 output pins to control the display.

ST_CP Storage register Clock

SH_CP Shift Register Clock both are positive edge triggered



DS Serial data In
OE output enable
MR Master Reset

The 8-bit serial-in shift register has its own input clock pin named SCK, while the D-Latch 8-bit registers use pin named RCK for transferring (latching) the 8-bit shift registers output to D-Latch output registers.

In normal operation according to the truth table above the 74HC595 shift registers clear pin (SCLR) should be put on logical high and the 8-bit D-Latch buffer output enable pin (G) should be put on logical low. By feeding the serial input pin (SER) with LPC 2129 master out slave in pin (MOSI) and connecting the master synchronous clock (SCK) to the 74HC595 shift registers clock (SCK), we could simply use the 74HC595 as the SPI slave device. Optionally we could connect the 74HC595 Q'H output pin (shift registers MSB bit) to the master in slave out pin (MISO); this optional connection will simply return the previous value of the shift registers to the SPI master register.

Data is written to the shift register serially, then latched onto the storage register. The storage register then controls 8 output lines. Data is shifted on the positive-going transitions of the shift register clock input.

The data in each register is transferred to the storage register on a positive-going transition of the storage register clock input. If both clocks are connected together, the shift register will always be one clock pulse ahead of the storage register. The shift register has a serial input and a serial standard output for cascading. It is also provided with asynchronous reset (active LOW) for all 8 shift register stages. The storage register has 8 parallel 3-state bus driver outputs. Data in the storage register appears at the output whenever the output enable input is LOW. The timing diagram below demonstrates how you would set the Q0-Q7 output pins to 11000011, assuming starting values

of 00000000.

ProgramCode

```
#include <LPC21XX.H>
#include "uart0.h"
#define HC595_CS 0x00000100
typedef unsigned int uint32;
typedef unsigned char uint8;
char status[] = "ELECTRONICS MAKER";
uint8 rcv_data;
const
DISP_TAB[16] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82,
0xF8, 0x80, 0x90, 0x88, 0x83, 0xC6, 0xA1, 0x86, 0x8E};

uint8 rcv_data;
////////////////////////// Delay
Function//////////////////////////
void delay(void) { unsigned volatile long
i,j;
for(i=0;i<10000;i++) for(j=0;j<30;j++) ;
}

//////////////////////////SPI
Initialize//////////////////////////
void spilnit(void)
{
SOSPCCR=0x52;
SOSPCR=0x30;
}

////////////////////////// Sending
DataSerially//////////////////////////
uint8 spiChar(uint8 data)
{
IOOCLR=HC595_CS;
SOSPDR=data;
while(0==(SOSPSR&0x80));
IOOSET=HC595_CS;
return(SOSPDR);
}

////////////////////////// Main
function//////////////////////////
int main(void)
{
uint8 i;
PINSELO = 0x00005505; IOODIR=HC595_CS;
0
spilnit();
uart0Init(); while(1)
{
for(i=0;i<16;i++)
{
if(i<10){putchar(i+'0');
}
else{putchar(i+7+'0');
}
}
rcv_data=spiChar(DISP_TAB[i]);
delay(); } serialPuts(status); }
}
```