SPI Basics:

- **Full Duplex**: Data sent and received simultaneously.
- **Master-Slave** architecture.
- Signals:
    - **SCK (Serial Clock)** – Clock provided by master.
    - **MOSI (Master Out Slave In)** – Data from master to slave.
    - **MISO (Master In Slave Out)** – Data from slave to master.
    - **SS (Slave Select)** – Active LOW, enables specific slave.

The LPC1768 uses **SSP** (Synchronous Serial Port) peripherals **to implement SPI communication**. In LPC1768, **SPI is implemented using SSP modules**, which are more **advanced** than basic SPI-only modules.
The **SSP peripheral** can support:

- SPI (what we use)
- TI synchronous serial interface
- Microwire interface

The LPC1768 microcontroller provides **three** SPI-compatible interfaces:

Peripheral	Interface Type	Notes
SSP0	SPI-compatible	Full SPI support
SSP1	SPI-compatible	Full SPI support
SPI (legacy)	Basic SPI interface	Less used, included for legacy compatibility

LPC1768 SPI-Compatible Peripherals and Pin Mappings

1. SPI Interface (Legacy SPI)

Function	Pin	PINSEL Register	PINSEL Bits
SCK	P0.15	PINSEL0	Bits 31:30 = 10
MISO	P0.17	PINSEL1	Bits 3:2 = 10
MOSI	P0.18	PINSEL1	Bits 5:4 = 10
SSEL (CS)	P0.16	PINSEL1	Bits 1:0 = 10

---

2. SSP0 (Recommended)

Function	Pin	PINSEL Register	PINSEL Bits
SCK	P0.15	PINSEL0	Bits 31:30 = 10
MISO	P0.17	PINSEL1	Bits 3:2 = 10
MOSI	P0.18	PINSEL1	Bits 5:4 = 10
SSEL	P0.16	PINSEL1	Bits 1:0 = 10

SSP0 **shares the same pins** as SPI, but offers **FIFO, better performance, and full-duplex**.

---

3. SSP1

Function	Pin Options	PINSEL Register	Bits
SCK	P0.7 or P1.20	PINSEL0 or PINSEL3	Bits 15:14 or Bits 9:8
MISO	P0.8 or P1.23	PINSEL0 or PINSEL3	Bits 17:16 or Bits 15:14
MOSI	P0.9 or P1.24	PINSEL0 or PINSEL3	Bits 19:18 or Bits 17:16
SSEL	P0.6 or P1.21	PINSEL0 or PINSEL3	Bits 13:12 or Bits 11:10

SSP1 provides **alternate pin options**, so it's useful if SSP0 pins are already in use.

Program:

1. Devices in SPI Communication (Master and Slave)

Role	Device	Description
Master	LPC1768 Microcontroller (SSP1 Peripheral)	Generates clock (SCK), selects slave (SS), sends and receives data
Slave	MAX7219 Display Driver IC	Receives SPI commands and controls the 7-segment displays accordingly
Output	7-Segment LED Displays (typically 4 or 8)	Connected to MAX7219’s output pins; shows numbers/characters as instructed

MAX7219 has two display modes:

Mode	Behavior
Decode Mode	MAX7219 automatically decodes the data you send (like 0x01 to display '1', 0x07 to display '7' etc.). It maps these to 7-segment patterns internally.
No Decode	You must manually send custom segment bit patterns (like 0b01001111) for each segment (a-g, dp).

int main() – starts with:

ssp_spi_config(); // SPI1 and pin setup

---

ssp_spi_config() — FULL REGISTER-WISE BREAKDOWN

---

1. Power Control for Peripherals (PCONP)

LPC_SC->PCON |= (1<<10); // Enable power to SSP1

PCONP (Power Control Register)

- Bit 10 – PCSSP1: When set to 1, **SSP1 block is powered on**.

Why? All peripherals in LPC1768 are turned off by default to save power. You must enable SSP1 before using it.

---

2. Peripheral Clock Selection (PCLKSEL0)

LPC_SC->PCLKSEL0 |= (3<<20); // Bits 21:20 → PCLK_SSP1 = CCLK/8

PCLKSEL0 controls the clock for peripherals like UART, SPI, etc.

- Bits 21:20 = 11 (binary) → Divide system clock (CCLK) by 8
  If CCLK = 4 MHz → PCLK_SSP1 = 500 kHz

Value	Meaning
00	CCLK / 4 (default)
01	CCLK
10	CCLK / 2
11	CCLK / 8 ✅

---

3. PINSEL0 (Pin Function Select Register)

```
LPC_PINCON->PINSEL0 |= (1<<15); // SCK1  (P0.7)
LPC_PINCON->PINSEL0 |= (1<<17); // MISO1 (P0.8)
LPC_PINCON->PINSEL0 |= (1<<19); // MOSI1 (P0.9)
```

PINSEL0 selects alternate functions for GPIO pins.

- P0.7 → SCK1 (bits 15:14 = 10)
- P0.8 → MISO1 (bits 17:16 = 10)
- P0.9 → MOSI1 (bits 19:18 = 10)

Each pin needs 2 bits:
Set **bit at +1**, Clear bit at **+0** for 10 (select function 2)

These three are SPI communication lines.

---

4. FIODIR (Fast GPIO Direction Register)

```
LPC_GPIO0->FIODIR |= (1<<6); // P0.6 as output → SS (Slave Select)
```

FIODIR sets the direction of GPIO pins.

- Bit 6 = 1 → output
- Used to **manually control SS** (Slave Select) for MAX7219

---

5. SSP1 Control Register 0 (CR0)

```
LPC_SSP1->CR0 |= 0x0B; // Data size = 12 bits
```

CR0 (Control Register 0) — Format, data size, polarity

Bits	Field	Description
3:0	DSS	Data size select (0x0B = 11 + 1 = 12 bits)
5:4	FRF	Frame Format (00 = SPI)
7:6		Clock polarity & phase (Mode 0)
15:8	SCR	Serial Clock Rate (SCR = 0)

---

6. SSP1 Prescaler Register (CPSR)

```
LPC_SSP1->CPSR = 20;
```

CPSR (Clock Prescale Register)

- SPI Clock = PCLK / (CPSDVSR × (SCR + 1))
- CPSDVSR = 20, SCR = 0
- SPI Clock = 500 kHz / 20 = **25 kHz**

---

7. SSP1 Control Register 1 (CR1)

```
LPC_SSP1->CR1 &= ~(1<<0); // Loopback disabled
LPC_SSP1->CR1 |=  (1<<1); // Enable SSP1 ☑
LPC_SSP1->CR1 &= ~(1<<2); // Master mode
```

CR1 (Control Register 1)

Bit	Name	Description
0	LBM	Loopback mode (0 = disabled)
1	SSE	SSP Enable (1 = enable) ✅
2	MS	Master/Slave (0 = Master) ✅

---

spi_data_write(uint16_t data)

1. FIOCLR (Clear GPIO Pin)

```
LPC_GPIO0->FIOCLR = (1<<6); // SS = 0
```

Sets **Slave Select low** to initiate communication.

---

2. SSP1 Status Register (SR)

```
while(!(LPC_SSP1->SR & (1<<1))) {} // TNF = 1?
```

SR (Status Register)

Bit	Name	Description
1	TNF	Transmit FIFO Not Full ✅
2	RNE	Receive FIFO Not Empty (not used here)
4	BSY	SSP Busy (1 = transmission in progress)

- This loop waits till **TX FIFO has space** to send data.

---

3. Data Register (DR)

```
LPC_SSP1->DR = data;
```

DR is **where you write your 12-bit SPI data**. This is sent out serially.

---

4. Wait till transfer finishes

```
while((LPC_SSP1->SR & (1<<4))) {} // BSY = 0?
```

- Bit 4: **BSY (Busy flag)** — wait till SPI transaction is complete.

---

5. FIOSET (Set GPIO Pin)

```
LPC_GPIO0->FIOSET = (1<<6); // SS = 1 → End transmission
```

- Sets **SS high** again → **deselect slave (MAX7219)**

---

delay(int32_t)

Simple nested for-loop for blocking delay.

Summary Table — Registers Used

Register	**Description**
PCONP	Enables SSP1 module
PCLKSEL0	Sets SSP1 peripheral clock
PINSEL0	Selects alternate function for SPI pins
FIODIR	Configures GPIO pin directions
FIOCLR/FIOSET	Controls Slave Select line
CR0 (SSP1)	Data size, frame format, mode
CR1 (SSP1)	Enables SSP, sets master/slave
CPSR (SSP1)	Clock prescale for SPI
SR (SSP1)	Status register: TX FIFO, busy flags
DR (SSP1)	Data register (write to send data)

There are two display modes in MAX7219:

1. **Decode Mode ON** – Only **0–9, E, H, L, P, - (dash)** are supported directly.
2. **Decode Mode OFF (No Decode)** – You must manually send **7-segment bit patterns** for each character (more flexible).

Decode Mode ON

This is set by:

```
spi_data_write(0x090F);  // Decode mode ON for digits 0 to 3
```

This tells MAX7219 to automatically decode data for digits 0–3 using an internal character map.

Supported Characters in Decode Mode

Data (HEX)	Character shown
0x0	0
0x1	1
0x2	2
0x3	3
0x4	4
0x5	5
0x6	6
0x7	7
0x8	8
0x9	9
0xA	- (dash)

Data (HEX)	Character shown
0xB	E
0xC	H
0xD	L
0xE	P
0xF	Blank (no segments)

Any other values (like 0xF → blank) will not show anything meaningful.

Example code:

```
spi_data_write(0x0101); // Display "1"

spi_data_write(0x0207); // Display "7"

spi_data_write(0x0306); // Display "6"

spi_data_write(0x0408); // Display "8"


spi_data_write(0x010C); // Display "H"

spi_data_write(0x020B); // Display "E"

spi_data_write(0x030D); // Display "L"

spi_data_write(0x040E); // Display "P"
```

II) MAX7219 in non-decode mode (i.e., custom segment values):

Bit	Segment
D7	DP (decimal point)
D6	a
D5	b
D4	c

Bit	Segment
D3	d
D2	e
D1	f
D0	g

Step 2: Writing Custom Characters (Decode Mode OFF)

To do this:

spi_data_write(0x0900); // Turn OFF decode mode

Then send custom segment pattern like:

spi_data_write(0x01 << 8 | 0x7E); // Send pattern to Digit 1

The value 0x7E turns ON the segments needed for a letter or number.

Step 3: Example Patterns for Characters

Char	Binary	Hex Value
0	01111110	0x7E
1	00001100	0x0C
2	10110110	0xB6
3	10011110	0x9E
4	11001100	0xCC
5	11011010	0xDA
6	11111010	0xFA
7	00001110	0x0E

Char	Binary	Hex Value
8	11111110	0xFE
9	11011110	0xDE
A	11101110	0xEE
b	11111000	0xF8
C	01110010	0x72
d	10111100	0xBC
E	11110010	0xF2
F	11100010	0xE2
-	10000000	0x80
Blank	00000000	0x00

You can make your own characters by turning ON the right segments (set their bits to 1).

Step 4: Sample Code for A, b, C, d

```
spi_data_write(0x0900);     // Disable decode mode

spi_data_write(0x01 << 8 | 0xEE); // 'A'

spi_data_write(0x02 << 8 | 0xF8); // 'b'

spi_data_write(0x03 << 8 | 0x72); // 'C'

spi_data_write(0x04 << 8 | 0xBC); // 'd'
```
