Agenda

- The C8051F38x family
- C8051F38x advantages
- C8051F38x enhancements
- Firmware portability
C8051F38x Family Features

- **High speed pipelined 8051 MCU core**
  - 48 MIPS operation
  - Up to 64K flash and 4K RAM

- **Flexible clocking**
  - Internal oscillator with ±0.25% accuracy supports all USB and UART modes
  - Low frequency oscillator for low power operation

- **USB function controller**
  - Integrated clock recovery requires no external crystal
  - Integrated transceiver requires no external resistors

- **Two UART and SMBus/I²C peripherals**

- **High performance analog**
  - 10-bit, 500 Ksps ADC
  - Integrated voltage reference (15 ppm)
  - Fast comparators

- **Integrated regulator**

- **Small 5 mm x 5 mm package**

- **-40 to +85 C operation**
### 12 new USB flash-based devices

<table>
<thead>
<tr>
<th>Ordering Part Number</th>
<th>C8051F380-GQ</th>
<th>C8051F381-GQ</th>
<th>C8051F381-GM</th>
<th>C8051F382-GQ</th>
<th>C8051F383-GQ</th>
<th>C8051F383-GM</th>
<th>C8051F384-GQ</th>
<th>C8051F385-GQ</th>
<th>C8051F385-GM</th>
<th>C8051F386-GQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIPS (Peak)</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Flash or EPROM Code Memory (Bytes)</td>
<td>64k</td>
<td>64k</td>
<td>64k</td>
<td>32k</td>
<td>32k</td>
<td>32k</td>
<td>64k</td>
<td>64k</td>
<td>64k</td>
<td>32k</td>
</tr>
<tr>
<td>RAM (Bytes)</td>
<td>4352</td>
<td>4352</td>
<td>4352</td>
<td>2304</td>
<td>2304</td>
<td>2304</td>
<td>4352</td>
<td>4352</td>
<td>4352</td>
<td>2304</td>
</tr>
<tr>
<td>Calibrated Internal 48 MHz Oscillator</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Internal Low Frequency Oscillator</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>USB with 1k Endpoint RAM</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Supply Voltage Regulator</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SMBus/I2C</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Enhanced SPI</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>UARTs</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Timers (16-bit)</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Programmable Counter Array</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Digital Port I/Os</td>
<td>40</td>
<td>25</td>
<td>25</td>
<td>40</td>
<td>25</td>
<td>25</td>
<td>40</td>
<td>25</td>
<td>25</td>
<td>40</td>
</tr>
<tr>
<td>10-bit 500ksps ADC</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Internal Voltage Reference</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Temperature Sensor</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Analog Comparator</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>External Memory Interface (EMIF)</td>
<td>Yes</td>
<td>—</td>
<td>—</td>
<td>Yes</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Yes</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Package</td>
<td>TQFP48</td>
<td>LQFP32</td>
<td>QFN32</td>
<td>TQFP48</td>
<td>LQFP32</td>
<td>QFN32</td>
<td>TQFP48</td>
<td>LQFP32</td>
<td>QFN32</td>
<td>TQFP48</td>
</tr>
</tbody>
</table>
C8051F38x USB Advantages

- Hardware implementation made simple with high functional density
  - Oscillators, resistors, voltage supply regulators and in system programmable memory are integrated on chip
  - All that is recommended are the USB ESD protection diodes

No external crystal, resistors, regulator or memory required
C8051F38x Family Enhancements (1 of 2)

- **Internal oscillator calibrated to 48 MHz**
  - Multiplier PLL not required
    - Multiplier SFR registers remain for backward compatibility with existing code base

- **More communications interfaces**
  - Adds second SMBus peripheral
    - SMBus peripherals are enhanced and provide hardware acknowledge and hardware address recognition
  - All devices have two UARTs
    - Second UART has its own baud rate generator and FIFO
C8051F38x Family Enhancements (2 of 2)

- **Low power optimization**
  - Voltage regulators can be disabled or placed into a low power state while maintaining voltage output
  - Pre-fetch engine can be disabled in the standby state to reduce power

- **More timing peripherals**
  - Six general purpose 16 bit timers

- **Analog performance enhanced**
  - ADC sample rate increased to 500 Ksps
  - Voltage reference provides more options
    - 1.2 V/2.4 V internal reference voltages available
    - Can use the internal 1.8 V regulator as well as VDD for the reference
Firmware Porting Considerations (1 of 2)

- Firmware functionality between the existing C8051F34x family and the C8051F38x family remains unchanged in the default state
  - All SFR mappings and functionality remain compatible
  - SFRs for removed peripherals remain, such as CLKMUL, for backward compatibility

- Clock multiplier no longer present
- No change to firmware initialization needed

USB Clock remains unchanged
New firmware can utilize new features of the C8051F38x family

- Increase ADC sensitivity using lower reference voltage
- Multiplier initialization no longer needed
- Can place regulators in low power modes
- Can place pre-fetch engine in a low power mode

<table>
<thead>
<tr>
<th>REG01CN Register</th>
<th>Bit 7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Reg0DIS</td>
<td>VBSTAT</td>
<td>VBPOL</td>
<td>REG0MD</td>
<td>STOPCF</td>
<td>Reserved</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Reset</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
C8051F38x Clocking
Clocking Options

- **Clock sources**
  - Flexible internal oscillator
    - Default clock after reset
    - Factory calibrated to 48 MHz
  - Low frequency oscillator at 80 KHz
  - External oscillator
    - Supports CMOS oscillators, crystals, RC networks and capacitors

- **USB clock can be sourced directly from the internal high frequency oscillator**
  - No external crystal required
USB Clock

- USB Clock multiplier is not required since the internal oscillator is calibrated to 48 MHz
- CLKMUL register still exists for backward compatibility with other USB MCUs

Legacy USB Clock Selection Mux

Is now the 48 MHz high frequency oscillator

Register setting remains the same as legacy devices
The USB Peripheral
USB Controller

- Complete Full/Low speed USB 2.0 compliant function controller
- Device only, cannot be a host
- Up to 8 Endpoints
- Integrated transceiver with clock recovery and configurable pull-up resistors
- 1 KB FIFO for Endpoint data transfers
- Serial Interface Engine (SIE) handles low level protocol in hardware
  - Error checking
  - Packet validity
- Control, Interrupt, Bulk and Isochronous transfers supported

### Supported Endpoints

<table>
<thead>
<tr>
<th>Endpoint</th>
<th>Associated Pipes</th>
<th>USB Protocol Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint0</td>
<td>Endpoint0 IN</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>Endpoint0 OUT</td>
<td>0x00</td>
</tr>
<tr>
<td>Endpoint1</td>
<td>Endpoint1 IN</td>
<td>0x81</td>
</tr>
<tr>
<td></td>
<td>Endpoint1 OUT</td>
<td>0x01</td>
</tr>
<tr>
<td>Endpoint2</td>
<td>Endpoint2 IN</td>
<td>0x82</td>
</tr>
<tr>
<td></td>
<td>Endpoint2 OUT</td>
<td>0x02</td>
</tr>
<tr>
<td>Endpoint3</td>
<td>Endpoint3 IN</td>
<td>0x83</td>
</tr>
<tr>
<td></td>
<td>Endpoint3 OUT</td>
<td>0x03</td>
</tr>
</tbody>
</table>
USB Register Access Scheme

- Two SFRs used to provide access to the configuration registers (USB0ADR and USB0DAT)
  - First set USB0ADR to define the USB register to access
  - Write/Read data to/from USB0DAT

- Endpoint Access via the Index register
  - Set USB0ADR to point to the Index register
  - Use USB0DAT to write the endpoint address desired into the index register
  - Switch USB0ADR to point to the Endpoint Control/Status registers
  - Use USB0DAT to write/read data to/from the endpoint registers

- USB Endpoint FIFOs accessed via the indexing scheme

USB clock must be active when accessing the USB0 Control/Status registers
Indirect Addressing Flow Chart

- Indirect register access

### Indirect Write Data Flow
1. Poll for BUSY USB0ADR
   - Bit 7 = '0'
2. USBAD already set?
   - Y: Load Target USB0 register into USBADR
   - N: Write Data to USB0DAT
3. Write Data to USB0DAT

### Indirect Read Data Flow
1. Poll for BUSY USB0ADR
   - Bit 7 = 0
2. USBAD already set?
   - Y: Load Target USB0 register into USBADR
   - N: Poll for BUSY USB0ADR
   - Bit 7 = 0
3. Auto-read enabled
   - Y: Read data from USB0DAT
   - N: Write '1' to BUSY bit in USB0ADR
4. Poll for BUSY USB0ADR
   - Bit 7 = 0
   - Y: Read data from USB0DAT
   - N: Auto-read enabled
USB0 FIFO Allocation

- 1024 Bytes of FIFO available to the USB endpoints allocated in XRAM space

- Endpoints 1-3 can be configured as IN, OUT or split mode with both IN and OUT endpoints
  - Split mode halves the FIFO size available for each endpoint

- Each endpoint can be double buffered
  - Half the memory is available for each transaction
  - Max. packet size is halved
    - Example, IN endpoint 1 double buffered provides 64 bytes for each IN transaction

- FIFO access indirectly addressed
C8051 Interrupt Vectors

- Single interrupt vector for all USB events
- 11 Interrupt sources can trigger an interrupt event
  - ISR needs to be parsed to determine which interrupt is pending

<table>
<thead>
<tr>
<th>Interrupt Source</th>
<th>Interrupt Vector</th>
<th>Priority Order</th>
<th>Pending Flag</th>
<th>Bit addressable?</th>
<th>Cleared by HW?</th>
<th>Enable Flag</th>
<th>Priority Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>0x0000</td>
<td>Top</td>
<td>None</td>
<td>N/A</td>
<td>N/A</td>
<td>Always Enabled</td>
<td>Always Highest</td>
</tr>
<tr>
<td>External Interrupt 0</td>
<td>0x0003</td>
<td>0</td>
<td>IE0 (TCON.1)</td>
<td>Y</td>
<td>Y</td>
<td>EX0 (IE.0)</td>
<td>PX0 (IP.0)</td>
</tr>
<tr>
<td>Timer 0 Overflow</td>
<td>0x000B</td>
<td>1</td>
<td>TF0 (TCON.5)</td>
<td>Y</td>
<td>Y</td>
<td>ETO (IE.1)</td>
<td>PT0 (IP.1)</td>
</tr>
<tr>
<td>External Interrupt 1</td>
<td>0x0013</td>
<td>2</td>
<td>IE1 (TCON.3)</td>
<td>Y</td>
<td>Y</td>
<td>EX1 (IE.2)</td>
<td>PX1 (IP.2)</td>
</tr>
<tr>
<td>Timer 1 Overflow</td>
<td>0x001B</td>
<td>3</td>
<td>TF1 (TCON.7)</td>
<td>Y</td>
<td>Y</td>
<td>ET1 (IE.3)</td>
<td>PT1 (IP.3)</td>
</tr>
<tr>
<td>UART0</td>
<td>0x0023</td>
<td>4</td>
<td>RI0 (SCON.0)</td>
<td>Y</td>
<td>N</td>
<td>ES0 (IE.4)</td>
<td>PS0 (IP.4)</td>
</tr>
<tr>
<td>Timer 2 Overflow</td>
<td>0x002B</td>
<td>5</td>
<td>TF2H (TMROCN.7)</td>
<td>Y</td>
<td>N</td>
<td>ET2 (IE.5)</td>
<td>PT2 (IP.5)</td>
</tr>
<tr>
<td>SPI0</td>
<td>0x0033</td>
<td>6</td>
<td>SPIF (SPIOCN.7)</td>
<td>Y</td>
<td>N</td>
<td>ESP0 (IE.6)</td>
<td>PSPI0 (IP.6)</td>
</tr>
<tr>
<td>SMB0</td>
<td>0x003B</td>
<td>7</td>
<td>SI (SMB0CN.0)</td>
<td>Y</td>
<td>N</td>
<td>ESMB0 (EIE1.0)</td>
<td>PSMB0 (EIP1.0)</td>
</tr>
<tr>
<td>USB0</td>
<td>0x0043</td>
<td>8</td>
<td>Special</td>
<td>N</td>
<td>N</td>
<td>EUSB0 (EIE1.1)</td>
<td>PUSB0 (EIP1.1)</td>
</tr>
<tr>
<td>ADC0 Window Compare</td>
<td>0x004B</td>
<td>9</td>
<td>AD0WINT (ADC0CN.3)</td>
<td>Y</td>
<td>N</td>
<td>EWADC0 (EIE1.2)</td>
<td>PWADC0 (EIP1.2)</td>
</tr>
<tr>
<td>ADC0 Conversion</td>
<td>0x0053</td>
<td>10</td>
<td>AD0INT (ADC0CN.5)</td>
<td>Y</td>
<td>N</td>
<td>EADC0 (EIE1.3)</td>
<td>PADC0 (EIP1.3)</td>
</tr>
<tr>
<td>Programmable Counter</td>
<td>0x005B</td>
<td>11</td>
<td>CF (PCA0CN.7)</td>
<td>Y</td>
<td>N</td>
<td>EPCA0 (EIE1.4)</td>
<td>PPCA0 (EIP1.4)</td>
</tr>
<tr>
<td>Comparator0</td>
<td>0x0063</td>
<td>12</td>
<td>CP0IF (CPT0CN.4)</td>
<td>N</td>
<td>N</td>
<td>ECP0 (EIE1.5)</td>
<td>PCP0 (EIP1.5)</td>
</tr>
<tr>
<td>Comparator1</td>
<td>0x006B</td>
<td>13</td>
<td>CP1IF (CPT1CN.4)</td>
<td>N</td>
<td>N</td>
<td>ECP1 (EIE1.6)</td>
<td>PCP1 (EIP1.6)</td>
</tr>
<tr>
<td>Timer 3 Overflow</td>
<td>0x0073</td>
<td>14</td>
<td>TF3H (TMROCN.7)</td>
<td>N</td>
<td>N</td>
<td>ET3 (EIE1.7)</td>
<td>PT3 (EIP1.7)</td>
</tr>
<tr>
<td>VBUS Level</td>
<td>0x007B</td>
<td>15</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>EVBUS (EIE2.0)</td>
<td>PVBUS (EIP2.0)</td>
</tr>
</tbody>
</table>
Serial Interface Engine (SIE) handles data communications to the host in hardware

- Handles the handshake between the endpoint and the host device
- Generates an interrupt when valid data packets received
- Will not interrupt the CPU when an error in transmission occurs
- Moves valid data to/from the endpoint FIFOs
- Firmware only needs to be concerned with the data transferred

<table>
<thead>
<tr>
<th>Token Packet format:</th>
<th>Field</th>
<th>PID</th>
<th>Address</th>
<th>Endpoint</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SOF Packet format:</th>
<th>Field</th>
<th>PID</th>
<th>Frame Number</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>8</td>
<td>11</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Packet format:</th>
<th>Field</th>
<th>PID</th>
<th>Data</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>8</td>
<td>0-1023</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Handshake Packet format:</th>
<th>Field</th>
<th>PID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

SIE handles error checking
SIE handles handshaking
Firmware interfaces
Control Transfer to Endpoint 0

Setup packet to Endpoint 0:
- Both IN and OUT directions
- Used when sending the USB Standard Requests

- CRC OK
- 8 data bytes transferred
- ACK transmitted

Host sends Setup Packet

Packet Valid?

- Y
  - Load data to Endpoint0 FIFO
  - Generate Interrupt
  - Set Index to 0x00 to read E0CSR
- N
  - Do nothing
  - Discard Data

OPRDY?

- Y
  - Unload FIFO
  - Decode transfer request
  - Set SOPRDY = 1
  - SOPRDY - Service OUT Packet
- N
  - Do other
  - Get data from OUT request
  - Load data for IN request
- **Host is requesting data**
  - Data phase of control transfers
  - Used when sending data for the USB standard requests

```
IN Packet to Endpoint 0

- Host sends IN Packet
  - Packet Valid?
    - Yes: Load data to Endpoint0 FIFO
      - Last Packet of data?
        - Yes: DATAEND = 1
        - No: INPRDY = 1
    - No: Send NAK
- INPRDY Set?
  - Yes: Send FIFO Data
  - No: Discard Data
- No response
```

**SIE controlled**
**Firmware control**
**Host is sending data**
- Data phase of control transfers
- Used when receiving data for standard requests

- Host sends out Packet
  - Packet Valid?
    - Y: Load data to Endpoint0 FIFO
      - Y: Generate Interrupt
      - N: Discard Data
    - N: No response
  - N: No response

- Unload FIFO
  - Last data byte received?
    - Y: Set DATAEND = 1
    - N: Do other
  - N: Set SOPRDY = 1

- SIE controlled
- Firmware control
The SMBus/I²C Peripheral
SMBus/I²C Peripheral

- Master/Slave byte-wise serial data transfers (can switch on-the-fly)
- Clock signal generation on SCL (Master Mode only) and SDA data synchronization
- Timeout/bus error recognition, as defined by the SMB0CF configuration register
- START/STOP timing, detection, and generation
- SMBus peripheral supports both software address decoding and hardware address decoding
  - Hardware address decoding
    - Hardware controls the ACK/NACK of the address and data bytes
    - The SMBus peripheral can support masters without clock stretching at 400 kHz (for I²C)
    - Hardware control means less code, less overhead and more CPU resources available
- Bus arbitration
- Status information
- Supports SCL Low Timeout and Bus Free Timeout detection
SMBus Transfer Modes

- The SMBus interface may be configured to operate as Master and/or a Slave

- At any particular time, it will be operating in one of the following four modes:
  - Master transmitter (write operation)
  - Master receiver (read operation)
  - Slave transmitter (read operation)
  - Slave receiver (write operation)

- Peripheral is in master mode any time a START is generated
  - Remains in Master mode until it loses an arbitration or generates a STOP

- SMBus interrupts are generated at the end of all SMBus byte frames:
  - Receiver:
    - The interrupt for an ACK occurs before the ACK with hardware ACK generation disabled
    - The interrupt for an ACK occurs after the ACK when hardware ACK generation is enabled
  - Transmitter:
    - Interrupts occur after the ACK
The SMBCS1–0 bits in SMB0CF select the SMBus clock source
- Overflows from Timer 0, Timer 1 or Timer 2 set the time-base
- Used only when operating as a Master or when the Bus Free Timeout detection is enabled
- Selected clock source may be shared by other peripherals so long as the timer is left running at all times.
  - Example, Timer 1 overflows may generate the SMBus and UART baud rates simultaneously

\[
T_{\text{HighMin}} = T_{\text{LowMin}} = \frac{1}{f_{\text{ClockSourceOverflow}}}
\]

\[
\text{BitRate} = \frac{f_{\text{ClockSourceOverflow}}}{3}
\]

Timer overflow rate determines high and low time and must conform to the standards as well as the requirements of the system, i.e. bus loading affects timing.

Actual bit rate of the peripheral

\(T_{\text{HIGH}}\) typically twice as large as \(T_{\text{LOW}}\) (SCL not driven or extended by another device)
The SMBus hardware has the capability to automatically recognize incoming slave addresses and send an ACK without software intervention

- SMBus Slave Address register
  - Programmed device address
  - Addresses are 7 bits

- SMBus Slave Address Mask Registers
  - A 1 in the bit position enables the comparison with the incoming address
  - A 0 in the bit position is treated as a don’t care

- Will recognize the General Call Address (0x00)

<table>
<thead>
<tr>
<th>Mask (SMB0ADM) = 0x7E</th>
<th>1 1 1 1 1 1 0 EHACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address (SMB0ADR) = 0x34</td>
<td>0 1 1 0 1 0 0 GC</td>
</tr>
</tbody>
</table>

- Accepted match = 0x34
  - 0 1 1 0 1 0 0 x

- Accepted match = 0x35
  - 0 1 1 0 1 0 1 x
### SMBus Configuration: SMB0CN Register

- **SMB0CN bits can be used to identify the transfer mode:**
  - MASTER
  - TXMODE
  - STA
  - STO

- **All bits combined define the firmware action to take**

  - **Example**: A master data or address byte was transmitted; ACK received.

    - SMB0CN = 1100001
      - MASTER = 1
      - TXMODE = 1
      - STA = 0
      - STO = 0
      - ACKRQ = 0
      - ARBLOST = 0
      - ACK = 1

    - Possible next action for firmware:
      - Load new data to SMB0DAT
      - Send Stop
      - Send repeated start

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>MASTER</td>
<td>SMBus Master/Slave Indicator. This read-only bit indicates when the SMBus is operating as a master.</td>
<td>0: SMBus operating in slave mode. 1: SMBus operating in master mode.</td>
<td>N/A</td>
</tr>
<tr>
<td>6</td>
<td>TXMODE</td>
<td>SMBus Transmit Mode Indicator. This read-only bit indicates when the SMBus is operating as a transmitter.</td>
<td>0: SMBus in Receiver Mode. 1: SMBus in Transmitter Mode.</td>
<td>N/A</td>
</tr>
<tr>
<td>5</td>
<td>STA</td>
<td>SMBus Start Flag.</td>
<td>0: No Start or repeated Start detected. 1: Start or repeated Start detected.</td>
<td>0: No Start generated. 1: When Configured as a Master, initiates a START or repeated START.</td>
</tr>
<tr>
<td>4</td>
<td>STO</td>
<td>SMBus Stop Flag.</td>
<td>0: No Stop condition detected. 1: Stop condition detected (if in Slave Mode) or pending (if in Master Mode).</td>
<td>0: No STOP condition is transmitted. 1: When configured as a Master, causes a STOP condition to be transmitted after the next ACK cycle.</td>
</tr>
<tr>
<td>3</td>
<td>ACKRQ</td>
<td>SMBus Acknowledge Request.</td>
<td>0: No Ack requested 1: ACK requested</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>ARBLOST</td>
<td>SMBus Arbitration Lost Indicator.</td>
<td>0: No arbitration error. 1: Arbitration Lost</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>ACK</td>
<td>SMBus Acknowledge.</td>
<td>0: NACK received. 1: ACK received.</td>
<td>0: Send NACK 1: Send ACK</td>
</tr>
<tr>
<td>0</td>
<td>SI</td>
<td>SMBus Interrupt Flag. This bit is set by hardware. SI must be cleared by software. While SI is set, SCL is held low and the SMBus is stalled.</td>
<td>0: No interrupt pending 1: Interrupt Pending</td>
<td>0: Clear interrupt, and initiate next state machine event. 1: Force interrupt.</td>
</tr>
</tbody>
</table>
Acknowledgement Handling

Software acknowledgement
- EHACK bit in register SMB0ADM is cleared to 0
- Firmware on the device must detect incoming slave addresses and ACK or NACK the slave address and incoming data bytes.
  - Receiver—writing the ACK bit defines the outgoing ACK value
  - Transmitter—reading the ACK bit indicates the value received during the last ACK cycle

Hardware acknowledgement
- EHACK bit in register SMB0ADM is set to 1
- Automatic slave address recognition and ACK generation is enabled in hardware
  - Receiver—the value currently specified by the ACK bit will be automatically sent on the bus during the ACK cycle of an incoming data byte
  - Transmitter—reading the ACK bit indicates the value received on the last ACK cycle

- Transmit mode always interrupts after the ACK/NAK
- Indicates a successful transfer
Write: Master Transmitter

- **Transmit mode always interrupts after the ACK/NAK**
  - First byte transfer the device is the master transmitter and interrupts after the ACK
  - The device then continues to be the transmitter and generates the interrupt regardless of the hardware acknowledgement bit (EHACK)

Master Write Interrupt Generation

Interruption points:
- Start
- Slave Address
- W
- A
- Data Byte
- A
- Data Byte
- A
- Stop

Interrupts generated with EHACK = 1

Interrupts generated with EHACK = 0

Received by SMBus slave
Transmitted by the SMBus slave
Read: Master Receiver

- Transmit mode always interrupts after the ACK/NAK
  - First byte transfer the device is the master transmitter and interrupts after the ACK
  - The device then becomes the receiver and generates the interrupt based on the hardware acknowledgement bit (EHACK)
    - EHACK = 1 then interrupts occur after the ACK/NAK
    - EHACK = 0 then interrupts occur before the ACK/NAK period and firmware must write the desired value to the ACK bit
First byte transfer the device is the slave receiver for the address and direction bit

- The device continues to be the receiver and generates the interrupt based on the hardware acknowledgement bit (EHACK)
  - EHACK = 1 then interrupts occur after the ACK/NAK
  - EHACK = 0 then interrupts occur before the ACK/NAK period and firmware must write the desired value to the ACK bit
Read: Slave Transmitter

- First byte transfer the device is the slave receiver for the address and direction bit
  - EHACK = 1 then interrupts occur after the ACK/NAK
  - EHACK = 0 then interrupts occur before the ACK/NAK period and firmware must write the desired value to the ACK bit

- The device then becomes the transmitter and generates the interrupt after the Acknowledgement bit (ACK)
SMBus/I²C Code Examples

- Code examples can be found in the Silicon Labs install directory
  - Silabs\MCU\Examples\C8051F38x
  - Master and slave implementations available

- Example initialization routines found in the examples directory

```c
void SMBus_Init (void)
{
    // Save the current SFRPAGE
    U8 SFRPAGE_save = SFRPAGE;
    SFRPAGE = LEGACY_PAGE;

    SMB0CF = 0x5D; // Use Timer1 overflows as SMBus clock
                   // source;
                   // Enable slave node;
                   // Enable setup & hold time
                   // extensions;
                   // Enable SMBus Free timeout detect;
                   // Enable SCI low timeout detect;

    SMB0CF |= 0x00; // Enable SMBus;
    SFRPAGE = SFRPAGE_save;
}
```

```c
void SMBus_Init (void)
{
    // Save the current SFRPAGE
    U8 SFRPAGE_save = SFRPAGE;
    SFRPAGE = LEGACY_PAGE;

    SMB0CF = 0x1D; // Use Timer1 overflows as SMBus clock
                   // source;
                   // Enable slave node;
                   // Enable setup & hold time
                   // extensions;
                   // Enable SMBus Free timeout detect;
                   // Enable SCI low timeout detect;

    SMB0CF |= 0x00; // Enable SMBus;
    SFRPAGE = SFRPAGE_save;
}
```
C8051F38x Enhanced UART
Additional UART Module

- Asynchronous full-duplex serial port
  - Dedicated Baud rate generator
  - Three byte FIFO for receiving characters

- Baud rates should be less than the system clock divided by 16

- Multi-processor mode available

- Odd, even, mark or space parity supported

Enhanced UART Block Diagram
The UART has several modes of operation, selectable using the SMOD register.

All modes enable asynchronous communications:
- 5, 6, 7, or 8-bit UART
- Extra 9TH bit for multi-processor communications
- Parity can be enabled or disabled
- Stop bit length can be changed

Parity bit can be enabled or disabled or the bit time can be used for an extra bit
Stop bit is programmable for 1, 1.5 or 2 bit times
Baud Rate Calculations

- The baud rate is generated by using the following equation:

\[
BaudRate = \frac{SYSCLK}{(65536 - (SBRLH1 : SBRLL1))} \times \frac{1}{2} \times \frac{1}{\text{Prescaler}}
\]

**Baud Rate Example:**

Desired baud rate = 57600 baud
Clock input to Timer 1 = System clock = 48 MHz

Changing above equation:

\[
SBRL = 65536 - \left( \frac{SYSCLK}{2 \times BaudRate \times \text{Prescaler}} \right)
\]

\[
SBRL = 65536 - \left( \frac{48MHz}{2 \times 57600 \times 1} \right)
\]

\[
SBRL = 65120
\]
Silicon Labs Tools for USB Development
Firmware Examples

- Installed as part of the Silicon labs IDE and found at
  - [http://www.silabs.com/PRODUCTS/MCU/Pages/SoftwareDownloads.aspx](http://www.silabs.com/PRODUCTS/MCU/Pages/SoftwareDownloads.aspx)

- Firmware template for HID applications can be used for custom applications

- USB examples provided (includes host and device software)
  - USB bulk—uses the bulk transfer method to illustrate USB
  - USB HID—includes firmware template as a starting point for custom firmware
    - HID blinky
    - HID to UART
    - HID mouse example
  - USB interrupt—examples highlight firmware that utilize the interrupt transfer

- Other examples
  - Mass Storage Device (MSD)
  - Human interface device w/boot loader
  - USB streaming audio/isochronous
USBXpress

- Allows the developer to implement a USB application without USB expertise
  - Royalty free, Windows certified device driver that can be customized and distributed
  - Microsoft Windows 2000, XP, Vista, 7 and WinCE are supported

- For use with USB MCUs as well as fixed function devices

- Host side API and drivers included
  - No host side driver development required
  - Drivers certified through Microsoft and can be customized and certified by the end user

- Firmware API included
  - Access to USBXpress libraries via the firmware API

- Details can be found in AN169: *USBXpress Programmers User Guide*
C8051F380DK Development Kit

- **C8051F380DK development kit**
  - Enables real-time code development and evaluation of the C8051F38x product family
  - Includes:
    - C8051F380 target board
    - Quick start guide
    - Integrated development environment (IDE)
    - USB debug adaptor
    - Wall power adaptor
    - USB cables and complete documentation

- **TOOLSTICK381DC**
  - Enables a quick development and evaluation of the C8051F381 USB MCU
  - Available for $9.90 USD (MSRP)

The C8051F380DK Development Kit is available for $99.00 USD (MSRP)