



SILICON LABS

[www.silabs.com](http://www.silabs.com)

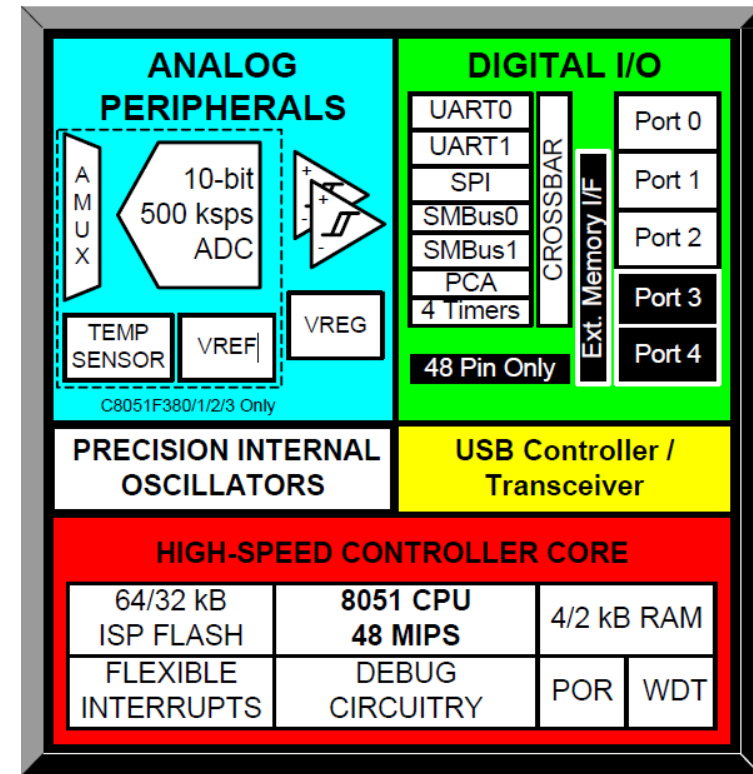
C8051F38x USB MCU

# 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  $\pm 0.25\%$  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/I<sup>2</sup>C 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**



*C8051F38x Block Diagram*

# C8051F38x Product Family

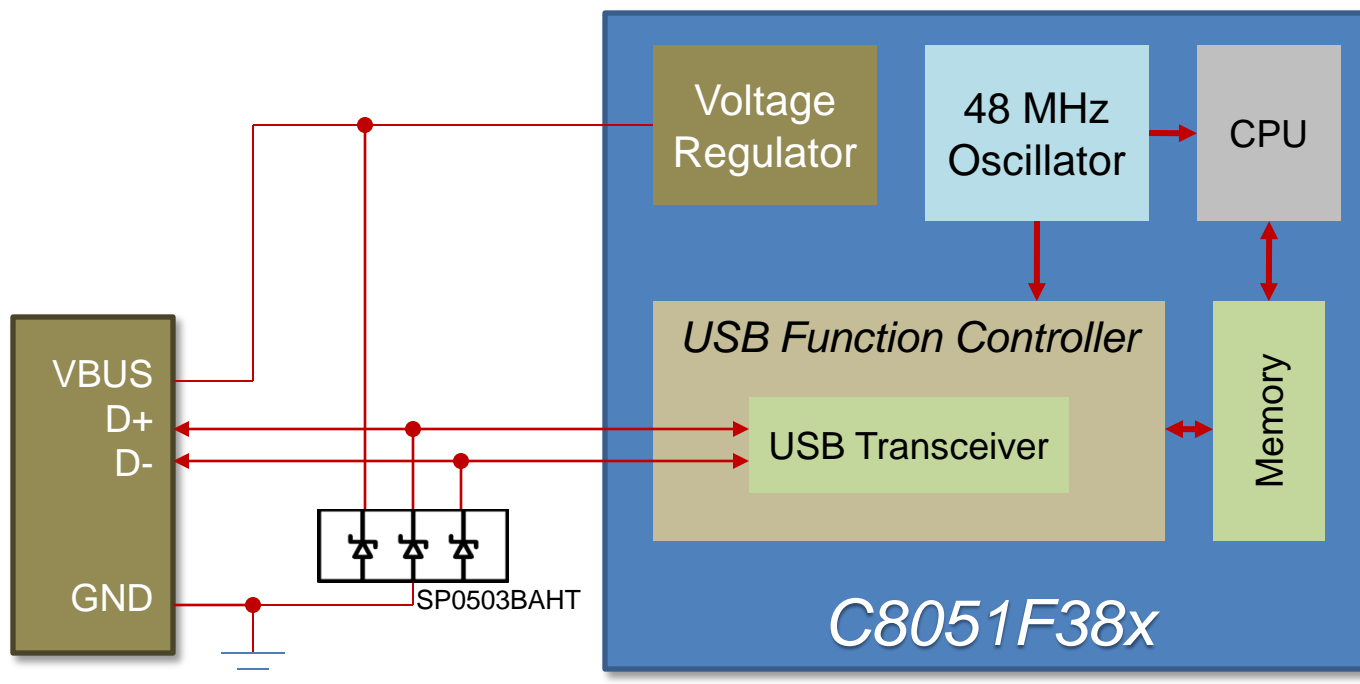
## ➤ 12 new USB flash-based devices

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

# 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

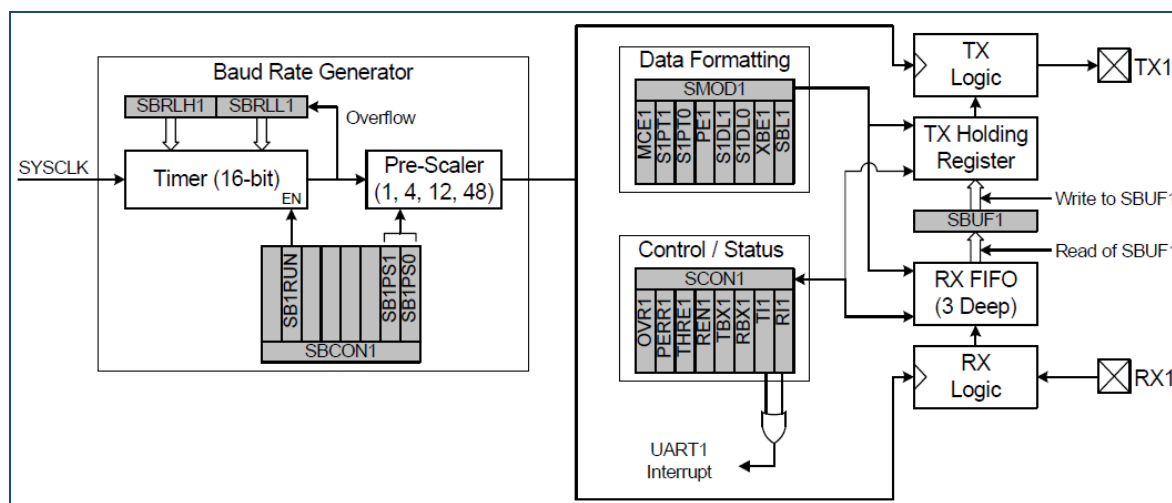
# C8051F38x Family Enhancements (1 of 2)

## ➤ 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 UART1 Peripheral

# C8051F38x Family Enhancements (2 of 2)

## ➤ Low power optimization

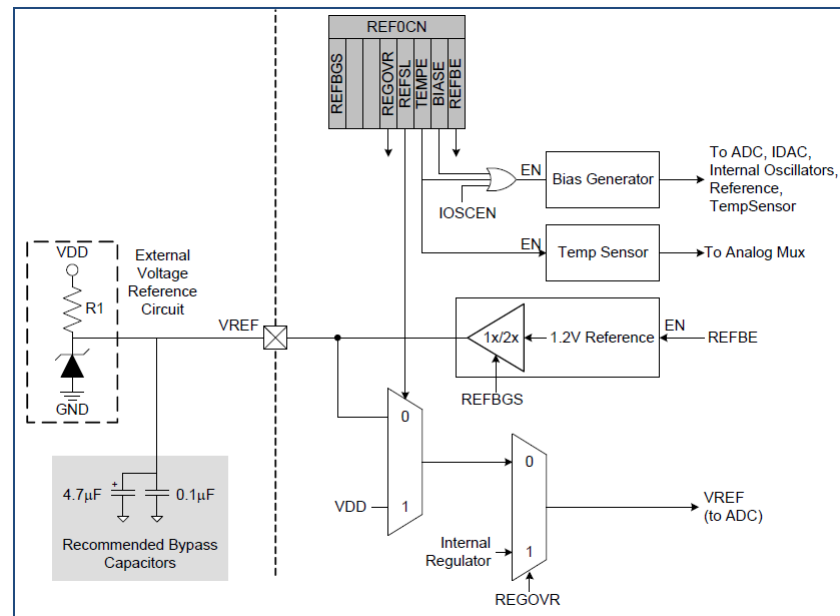
- Voltage regulators can be disabled or placed into a low power state while maintaining voltage output
- Pre-fetch engine can be disabled in the standby state to reduce power

## ➤ More timing peripherals

- Six general purpose 16 bit timers

## ➤ Analog performance enhanced

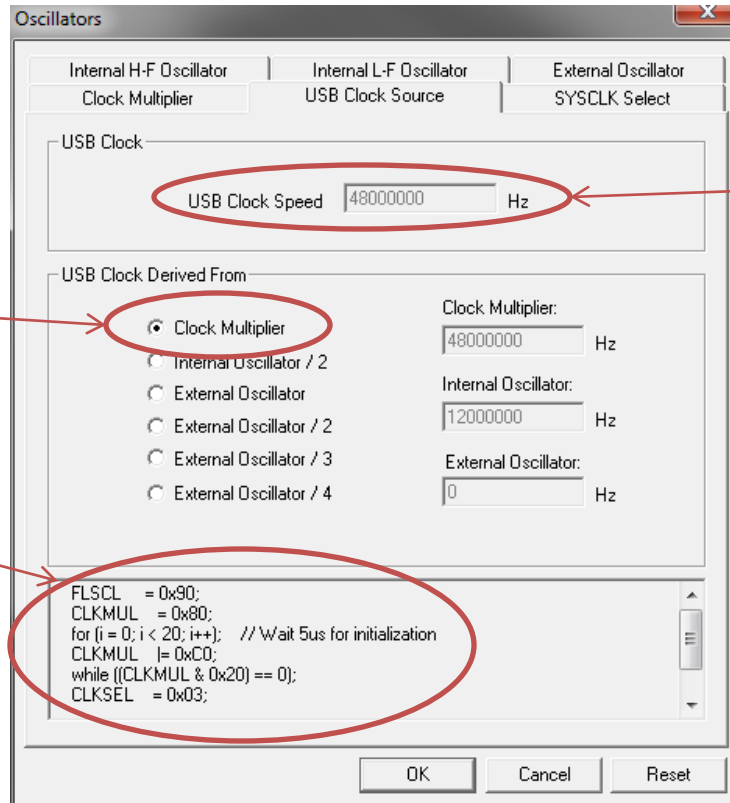
- ADC sample rate increased to 500 Ksps
- Voltage reference provides more options
  - 1.2 V/2.4 V internal reference voltages available
  - Can use the internal 1.8 V regulator as well as VDD for the reference



C8051F38x  $V_{REF}$  Peripheral

# 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

Configuration Wizard



# Firmware Porting Considerations (2 of 2)

## ➤ 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

Low power mode bits for regulators

*REG01CN Register*

Bit	7	6	5	4	3	2	1	0
Name	Reg0DIS	VBSTAT	VBPOL	REG0MD	STOPCF	Reserved	REG1MD	Reserved
Type	R/W	R	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0



SILICON LABS

[www.silabs.com](http://www.silabs.com)

# C8051F38x Clocking

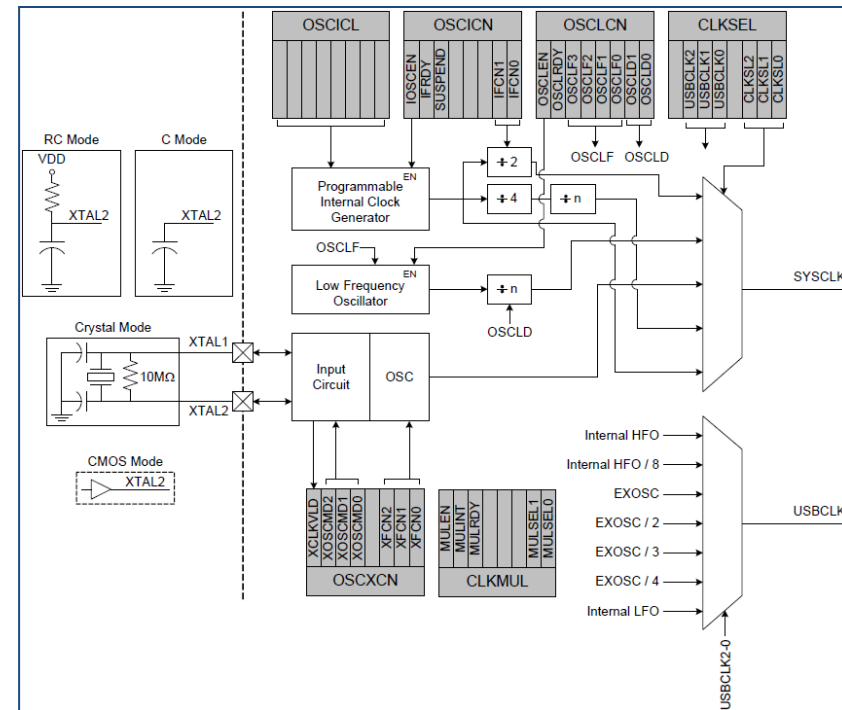
# Clocking Options

## ➤ 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

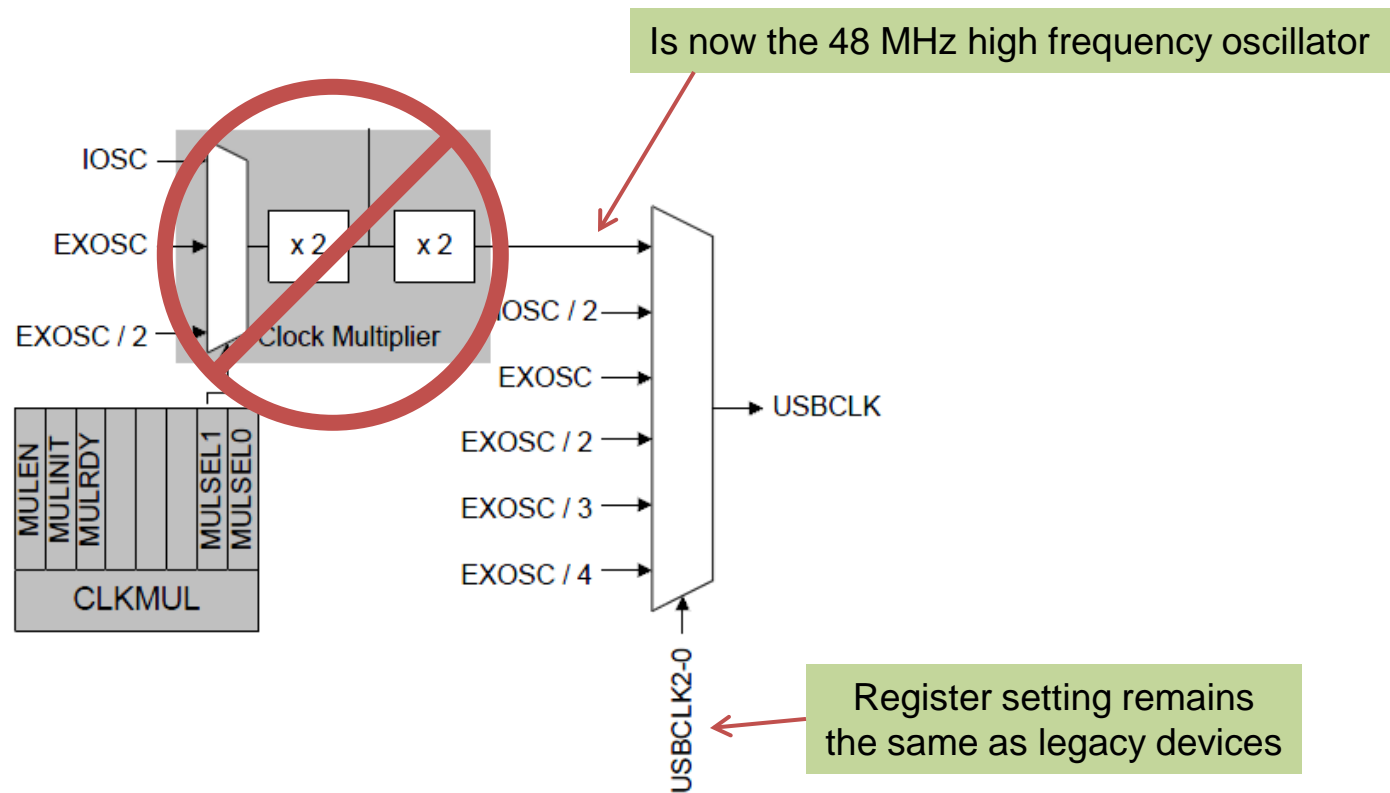
- No external crystal required



Oscillator Options

# 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



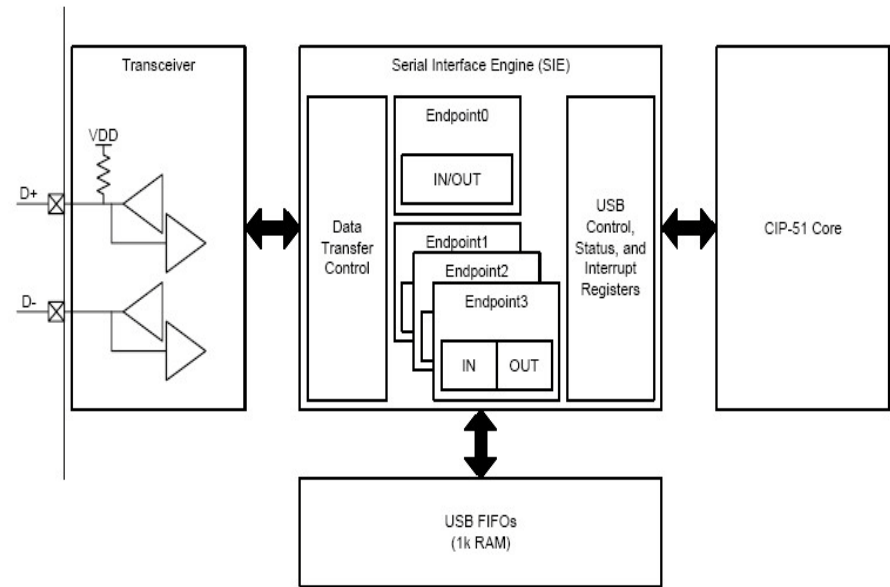
SILICON LABS

[www.silabs.com](http://www.silabs.com)

# 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**



*USB0 Peripheral*

Endpoint	Associated Pipes	USB Protocol Address
Endpoint0	Endpoint0 IN	0x00
	Endpoint0 OUT	0x00
Endpoint1	Endpoint1 IN	0x81
	Endpoint1 OUT	0x01
Endpoint2	Endpoint2 IN	0x82
	Endpoint2 OUT	0x02
Endpoint3	Endpoint3 IN	0x83
	Endpoint3 OUT	0x03

*Supported Endpoints*

# USB Register Access Scheme

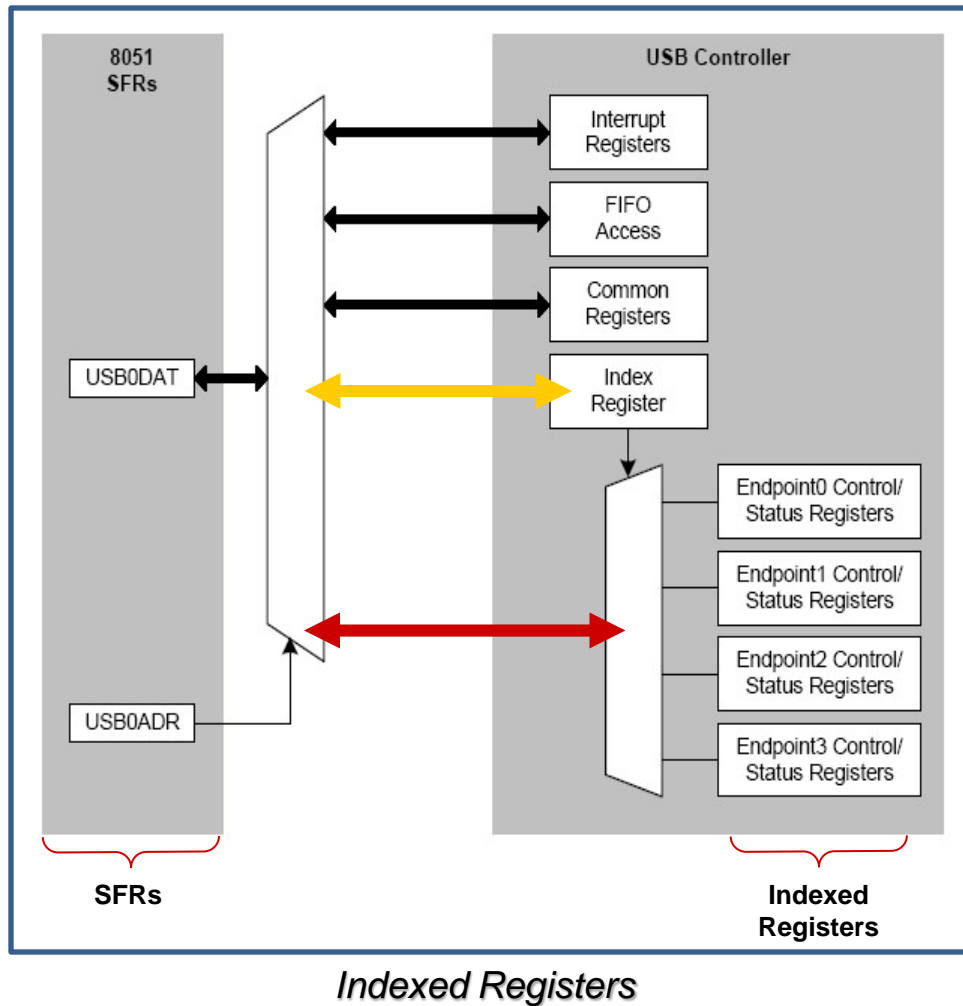
## ➤ Two SFRs used to provide access to the configuration registers (USB0ADR and USB0DAT)

- 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

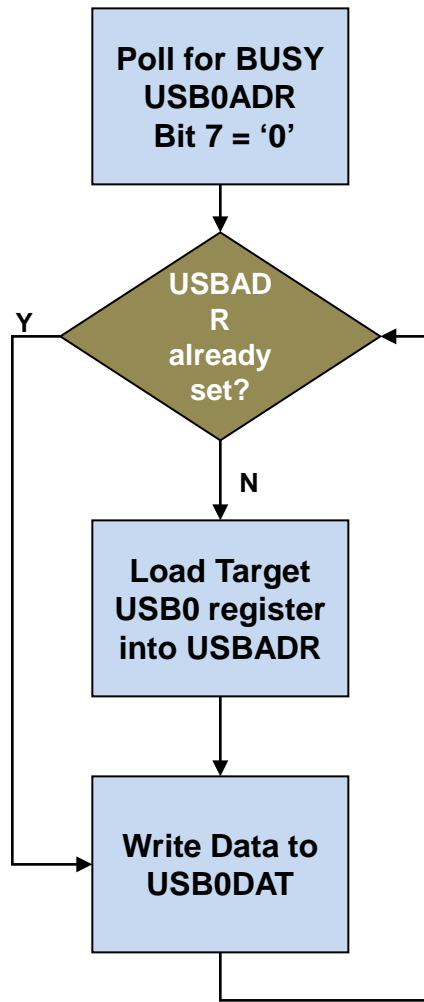
## ➤ USB Endpoint FIFOs accessed via the indexing scheme



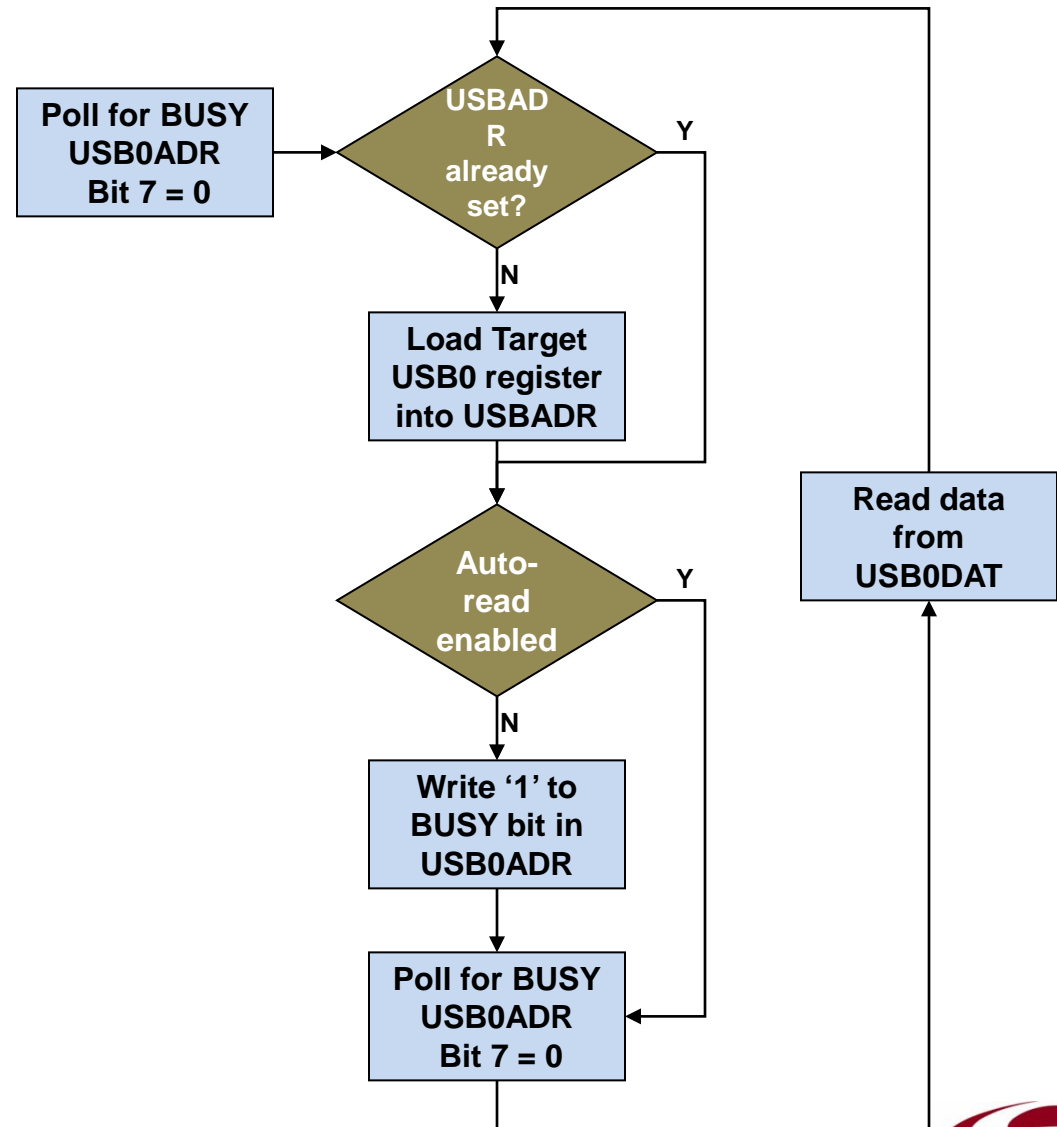
USB clock must be active when accessing the USB0 Control/Status registers

# Indirect Addressing Flow Chart

## ➤ Indirect register access



Indirect Write Data Flow



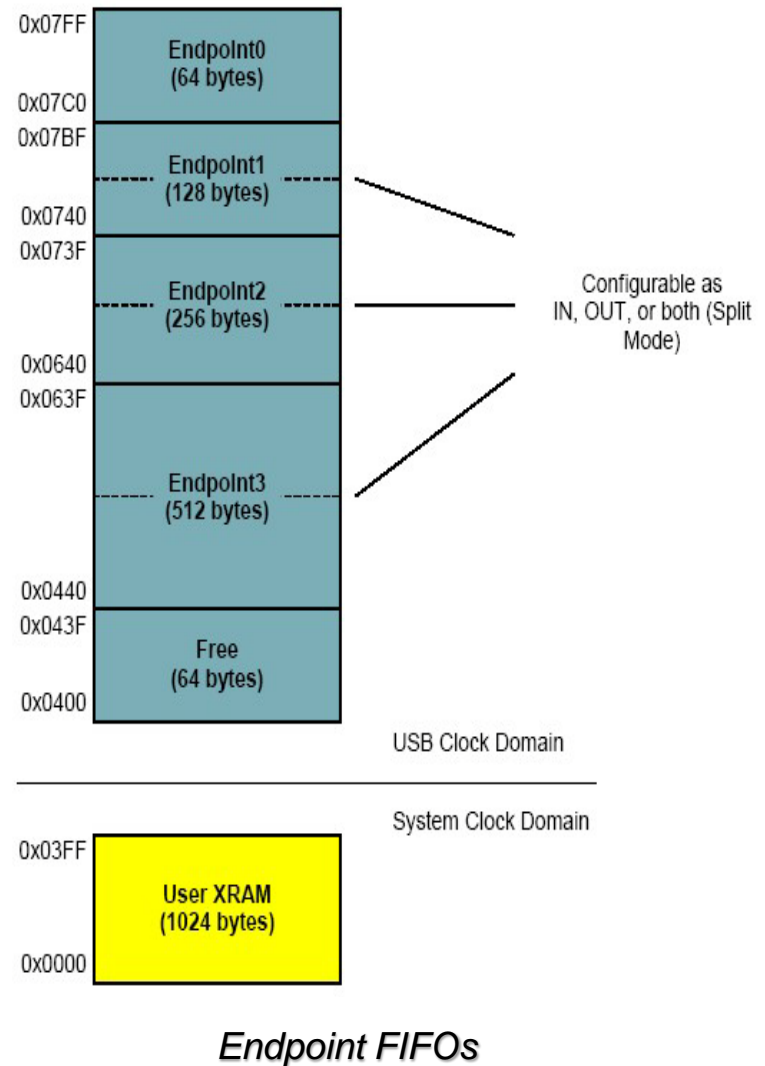
Indirect Read Data Flow



# 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**

IN/OUT Endpoint FIFO	USB Address
0	0x20
1	0x21
2	0x22
3	0x23



# C8051 Interrupt Vectors

➤ **Single interrupt vector for all USB events**

➤ **11 Interrupt sources can trigger an interrupt event**

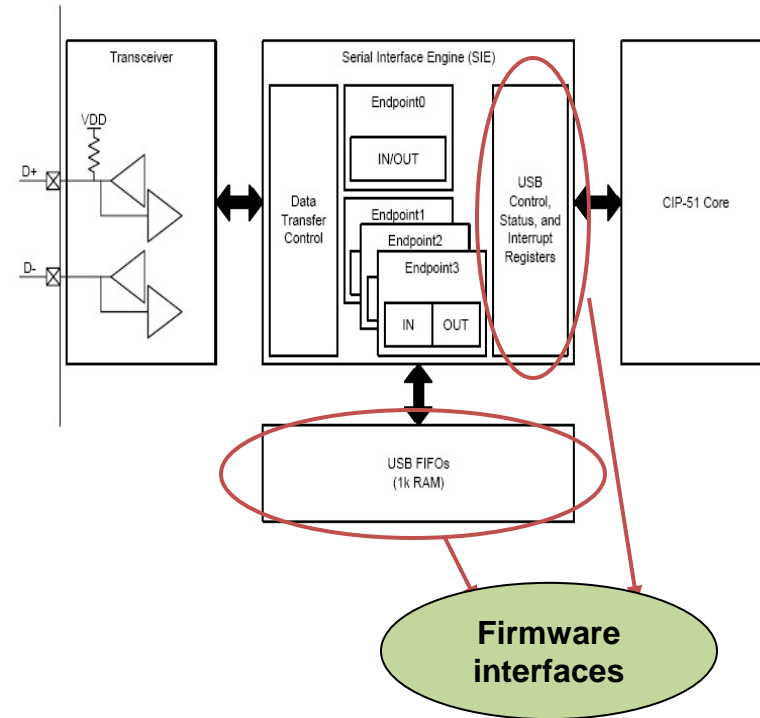
- ISR needs to be parsed to determine which interrupt is pending

Interrupt Source	Interrupt Vector	Priority Order	Pending Flag	Bit addressable?	Cleared by HW?	Enable Flag	Priority Control
Reset	0x0000	Top	None	N/A	N/A	Always Enabled	Always Highest
External Interrupt 0 (/INT0)	0x0003	0	IE0 (TCON.1)	Y	Y	EX0 (IE.0)	PX0 (IP.0)
Timer 0 Overflow	0x000B	1	TF0 (TCON.5)	Y	Y	ET0 (IE.1)	PT0 (IP.1)
External Interrupt 1 (/INT1)	0x0013	2	IE1 (TCON.3)	Y	Y	EX1 (IE.2)	PX1 (IP.2)
Timer 1 Overflow	0x001B	3	TF1 (TCON.7)	Y	Y	ET1 (IE.3)	PT1 (IP.3)
UART0	0x0023	4	RI0 (SCON0.0) TI0 (SCON0.1)	Y	N	ES0 (IE.4)	PS0 (IP.4)
Timer 2 Overflow	0x002B	5	TF2H (TMR2CN.7) TF2L (TMR2CN.6)	Y	N	ET2 (IE.5)	PT2 (IP.5)
SPI0	0x0033	6	SPIF (SPI0CN.7) WCOL (SPI0CN.6) MODF (SPI0CN.5) RXOVRN (SPI0CN.4)	Y	N	ESPI0 (IE.6)	PSPI0 (IP.6)
SMB0	0x003B	7	SI (SMB0CN.0)	Y	N	ESMB0 (EIE1.0)	PSMB0 (EIP1.0)
USB0	0x0043	8	Special	N	N	EUSB0 (EIE1.1)	PUSB0 (EIP1.1)
ADC0 Window Compare	0x004B	9	AD0WINT (ADC0CN.3)	Y	N	EWADC0 (EIE1.2)	PWADC0 (EIP1.2)
ADC0 Conversion Complete	0x0053	10	AD0INT (ADC0CN.5)	Y	N	EADC0 (EIE1.3)	PADC0 (EIP1.3)
Programmable Counter Array	0x005B	11	CF (PCA0CN.7) CCFn (PCA0CN.n)	Y	N	EPCA0 (EIE1.4)	PPCA0 (EIP1.4)
Comparator0	0x0063	12	CP0FIF (CPT0CN.4) CP0RIF (CPT0CN.5)	N	N	ECP0 (EIE1.5)	PCP0 (EIP1.5)
Comparator1	0x006B	13	CP1FIF (CPT1CN.4) CP1RIF (CPT1CN.5)	N	N	ECP1 (EIE1.6)	PCP1 (EIP1.6)
Timer 3 Overflow	0x0073	14	TF3H (TMR3CN.7) TF3L (TMR3CN.6)	N	N	ET3 (EIE1.7)	PT3 (EIP1.7)
VBUS Level	0x007B	15	N/A	N/A	N/A	EVBUS (EIE2.0)	PVBUS (EIP2.0)

# Serial Interface Engine (SIE)

## 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



Token Packet format:

Field	PID	Address	Endpoint	CRC
Bits	8	7	4	5

SOF Packet format:

Field	PID	Frame Number	CRC
Bits	8	11	5

Data Packet format:

Field	PID	Data	CRC
Bits	8	0-1023	16

Handshake Packet format:

Field	PID
Bits	8

SIE handles error checking

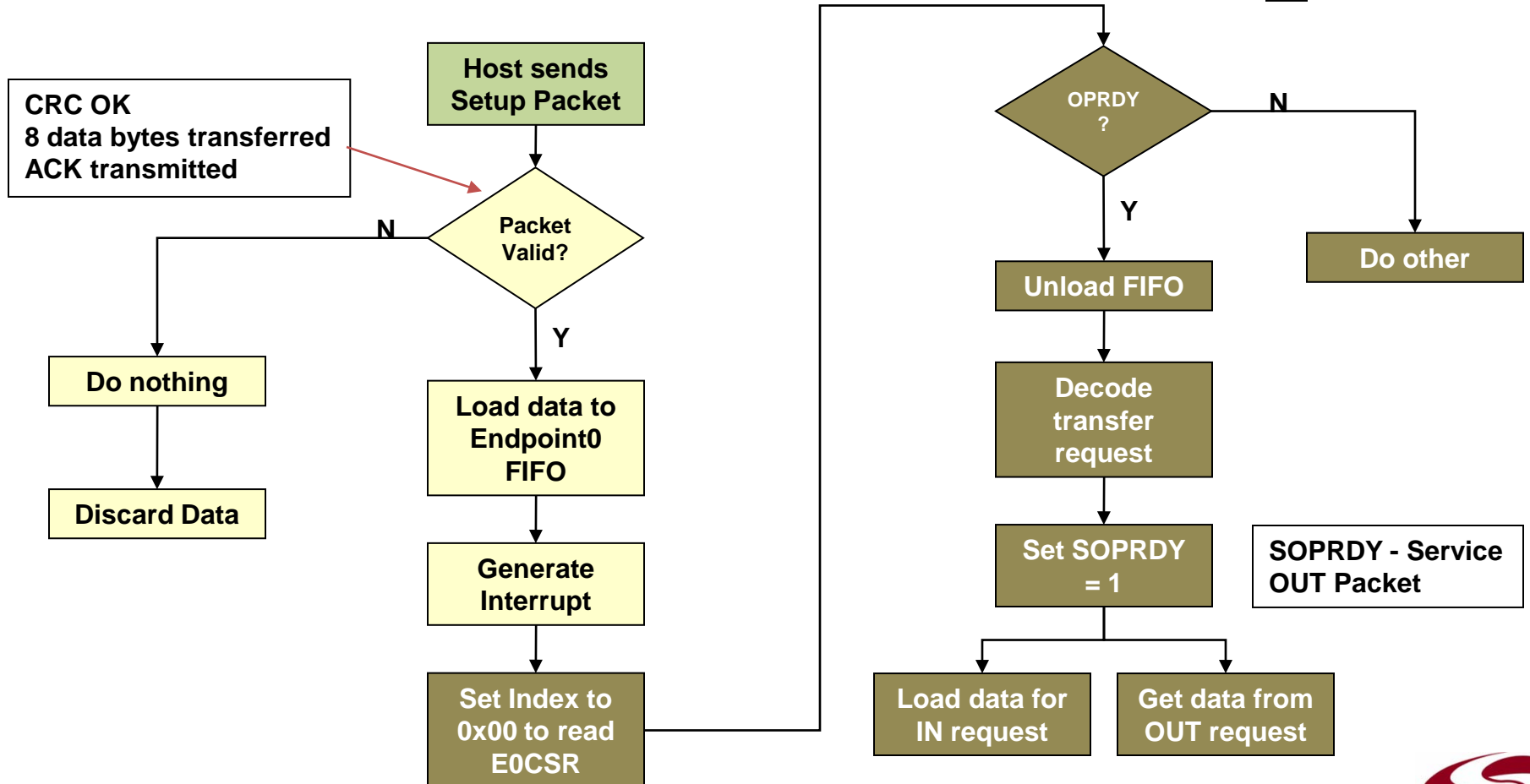
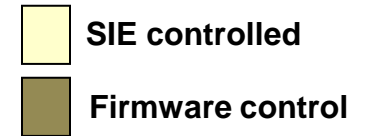
SIE handles handshaking

ACK  
NAK

# Control Transfer to Endpoint 0

## ➤ Setup packet to Endpoint 0:

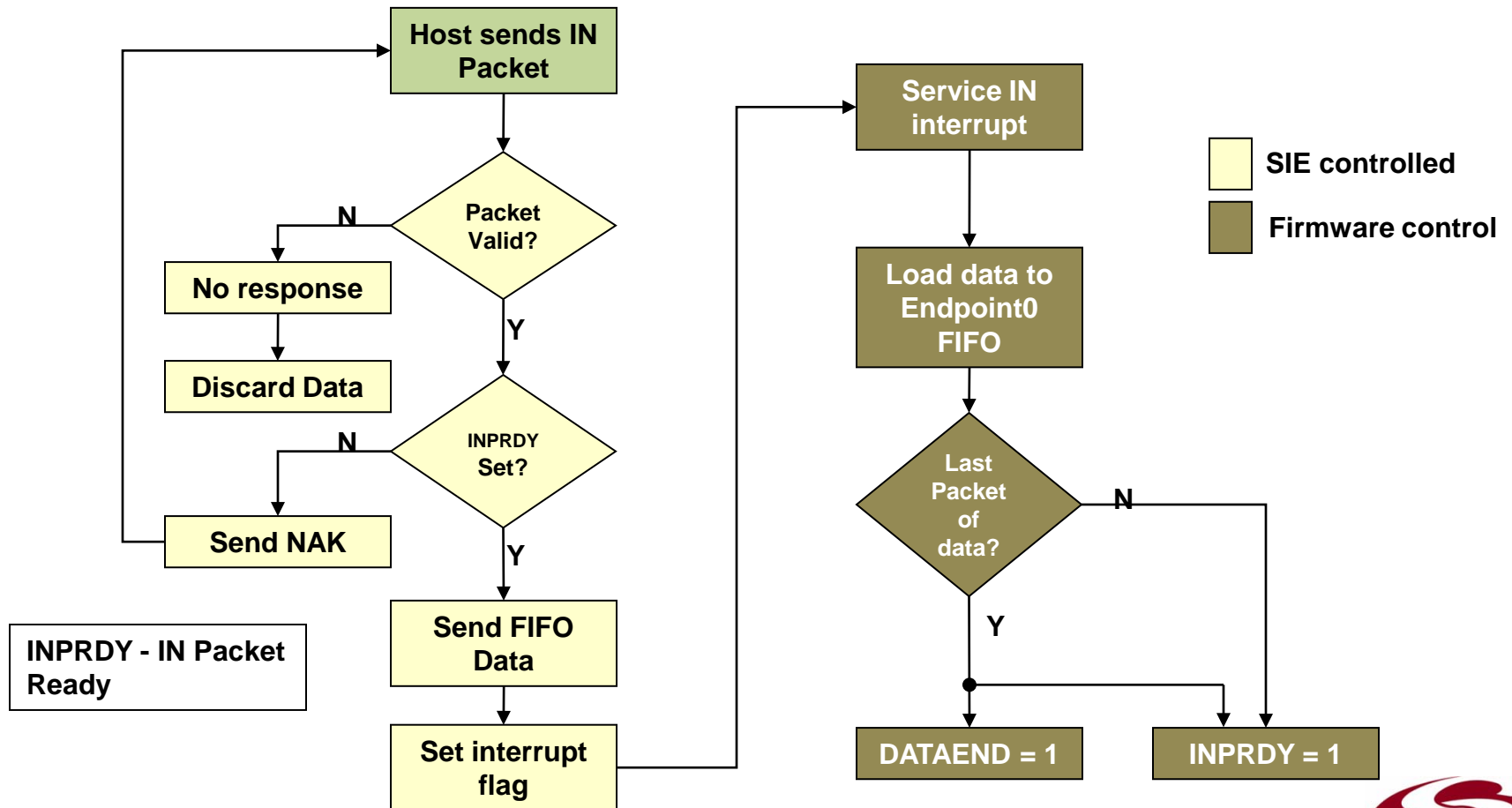
- Both IN and OUT directions
- Used when sending the USB Standard Requests



# IN Packet to Endpoint 0

## ➤ Host is requesting data

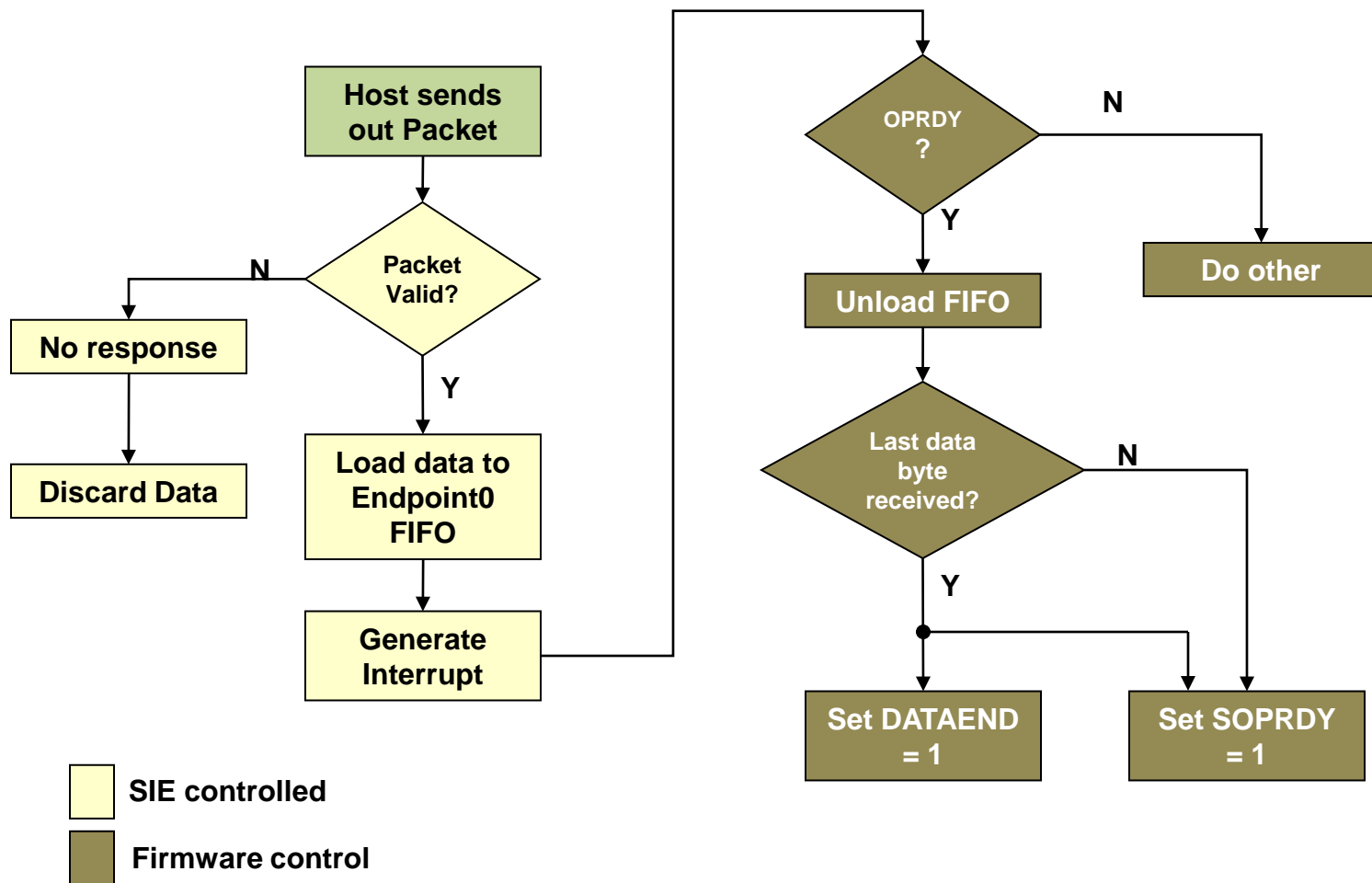
- Data phase of control transfers
- Used when sending data for the USB standard requests



# OUT Packet to Endpoint 0

## ➤ Host is sending data

- Data phase of control transfers
- Used when receiving data for standard requests





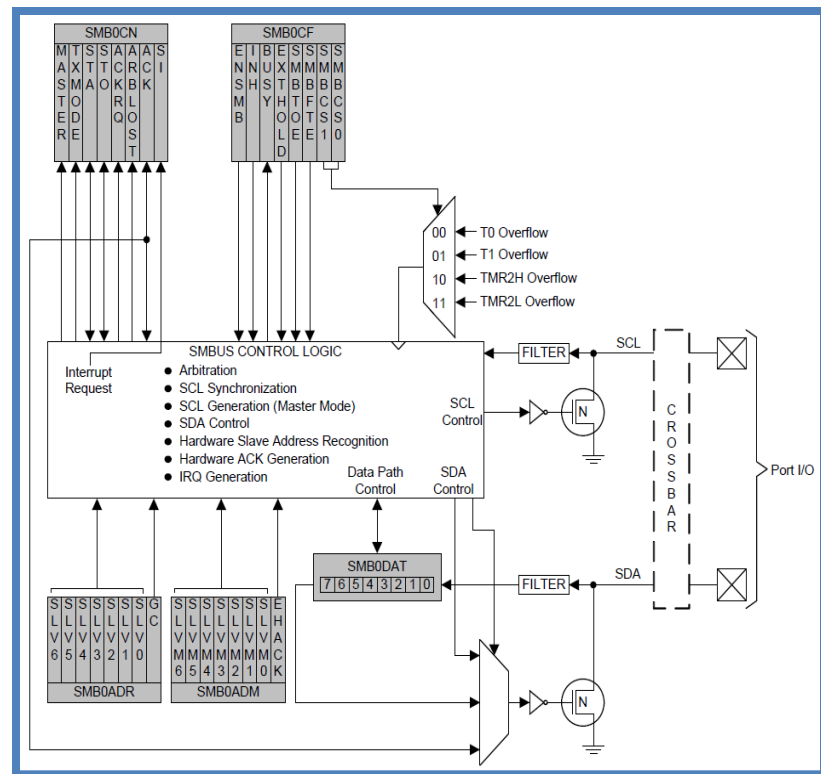
SILICON LABS

[www.silabs.com](http://www.silabs.com)

# The SMBus/I<sup>2</sup>C Peripheral

# SMBus/I<sup>2</sup>C 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 I<sup>2</sup>C)
    - 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

# SMBus Timing Control

## ➤ The **SMBCS1–0** bits in **SMB0CF** select the **SMBus clock source**

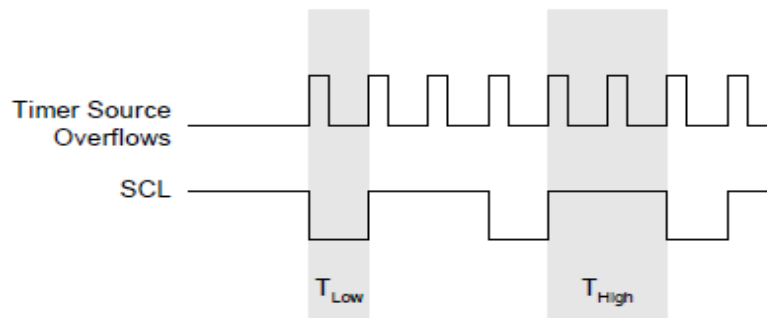
- Overflows from Timer 0, Timer 1 or Timer 2 set the time-base
- Used only when operating as a Master or when the Bus Free Timeout detection is enabled
- Selected clock source may be shared by other peripherals so long as the timer is left running at all times.
  - Example, Timer 1 overflows may generate the SMBus and UART baud rates simultaneously

$$T_{HighMin} = T_{LowMin} = \frac{1}{f_{ClockSourceOverflow}}$$

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.

$$BitRate = \frac{f_{ClockSourceOverflow}}{3}$$

Actual bit rate of the peripheral

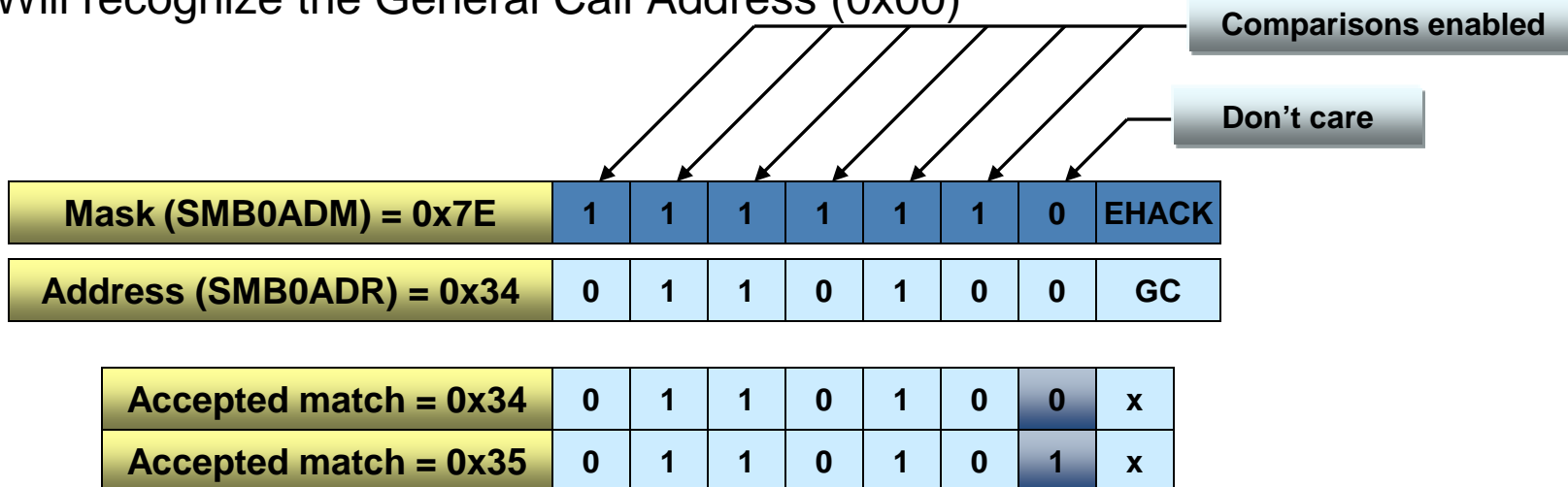


$T_{High}$  typically twice as large as  $T_{Low}$  (SCL not driven or extended by another device)

# SMBus Addressing

➤ **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)



# 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

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

# 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

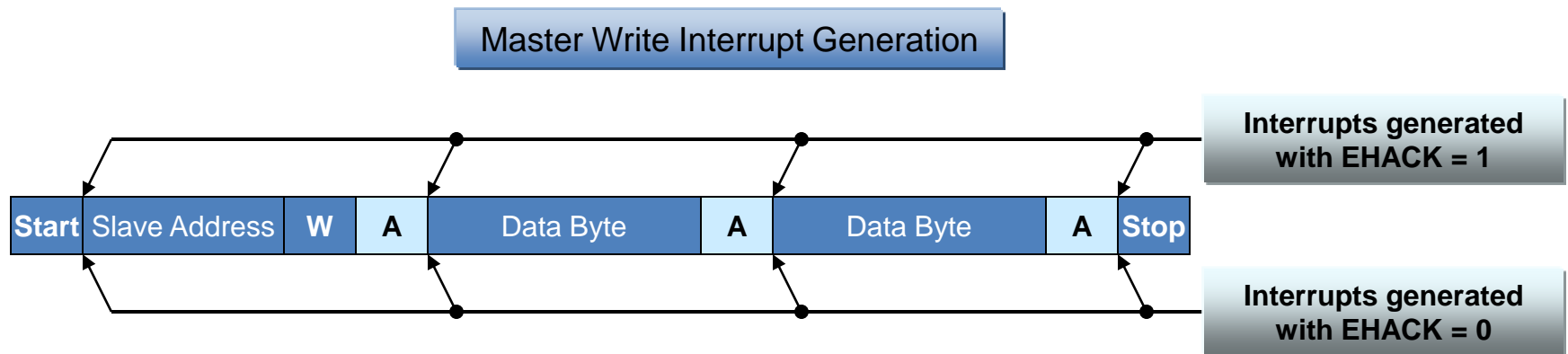
## ➤ **Hardware acknowledgement**

- 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)

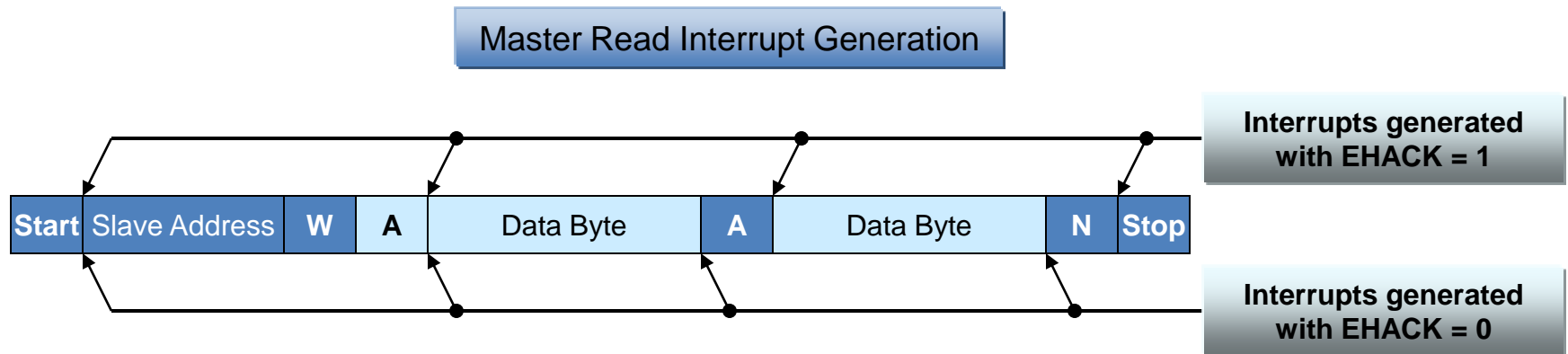


Received by SMBus slave

Transmitted by the SMBus slave

# Read: Master Receiver

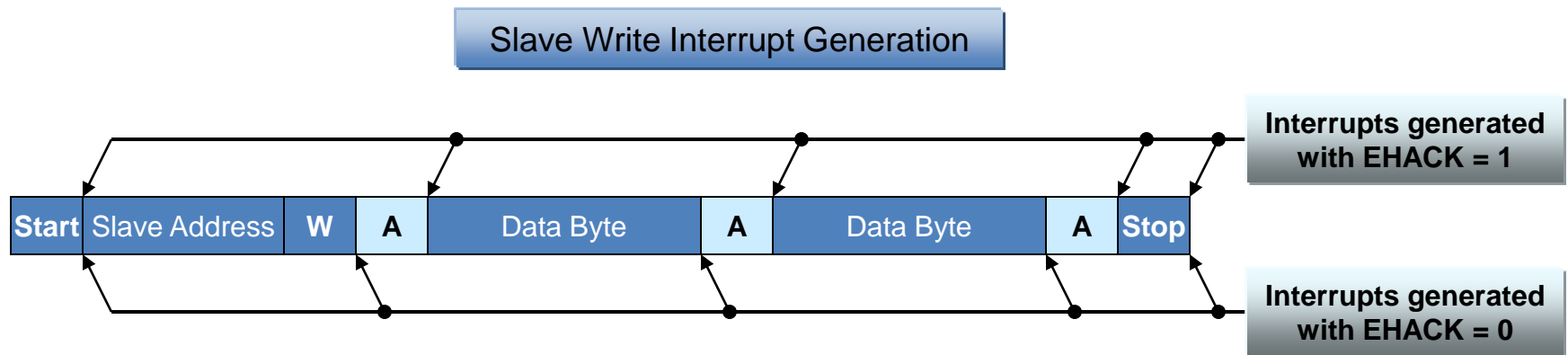
- ◆ Transmit mode always interrupts after the ACK/NAK
  - First byte transfer the device is the master transmitter and interrupts after the ACK
  - The device then becomes the receiver and generates the interrupt based on the hardware acknowledgement bit (EHACK)
    - EHACK = 1 then interrupts occur 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



Received by SMBus Slave  
Transmitted by the SMBus Slave

# Write: Slave Receiver

- **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

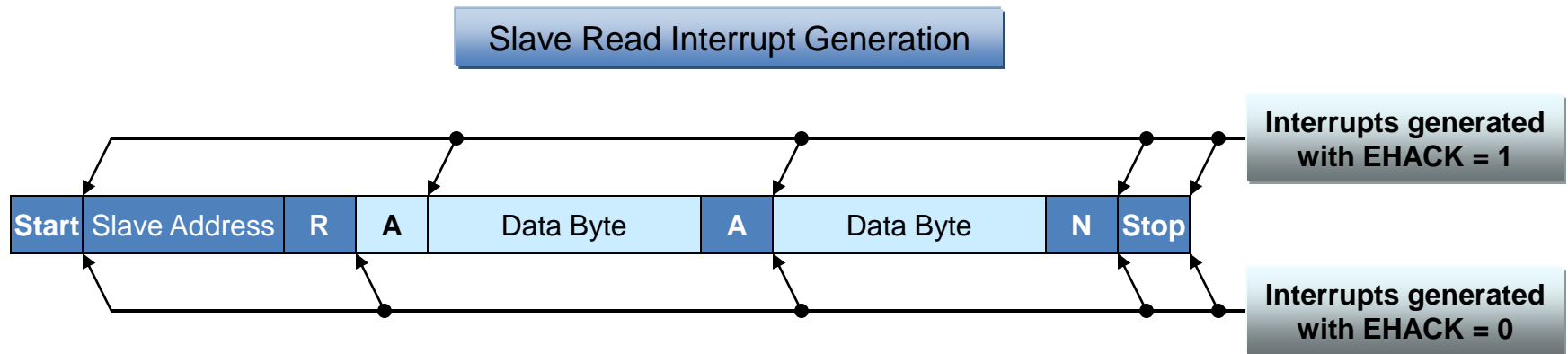


Received by SMBus Slave
Transmitted by the SMBus Slave



# 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)**



Received by SMBus Slave  
Transmitted by the SMBus Slave

# SMBus/I<sup>2</sup>C 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

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

    SFRPAGE = LEGACY_PAGE;

    SMBOCF = 0x5D;

    // Use Timer1 overflows as SMBus clock
    // source;
    // Disable slave mode;
    // Enable setup & hold time
    // extensions;
    // Enable SMBus Free timeout detect;
    // Enable SCL low timeout detect;

    SMBOCF |= 0x80;

    // Enable SMBus;

    SFRPAGE = SFRPAGE_save;
}
```

*Master*

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

    SFRPAGE = LEGACY_PAGE;

    SMBOCF = 0x1D;

    // Use Timer1 overflows as SMBus clock
    // source;
    // Enable slave mode;
    // Enable setup & hold time
    // extensions;
    // Enable SMBus Free timeout detect;
    // Enable SCL low timeout detect;

    SMBOCF |= 0x80;

    // Enable SMBus;

    SFRPAGE = SFRPAGE_save;
}
```

*Slave*



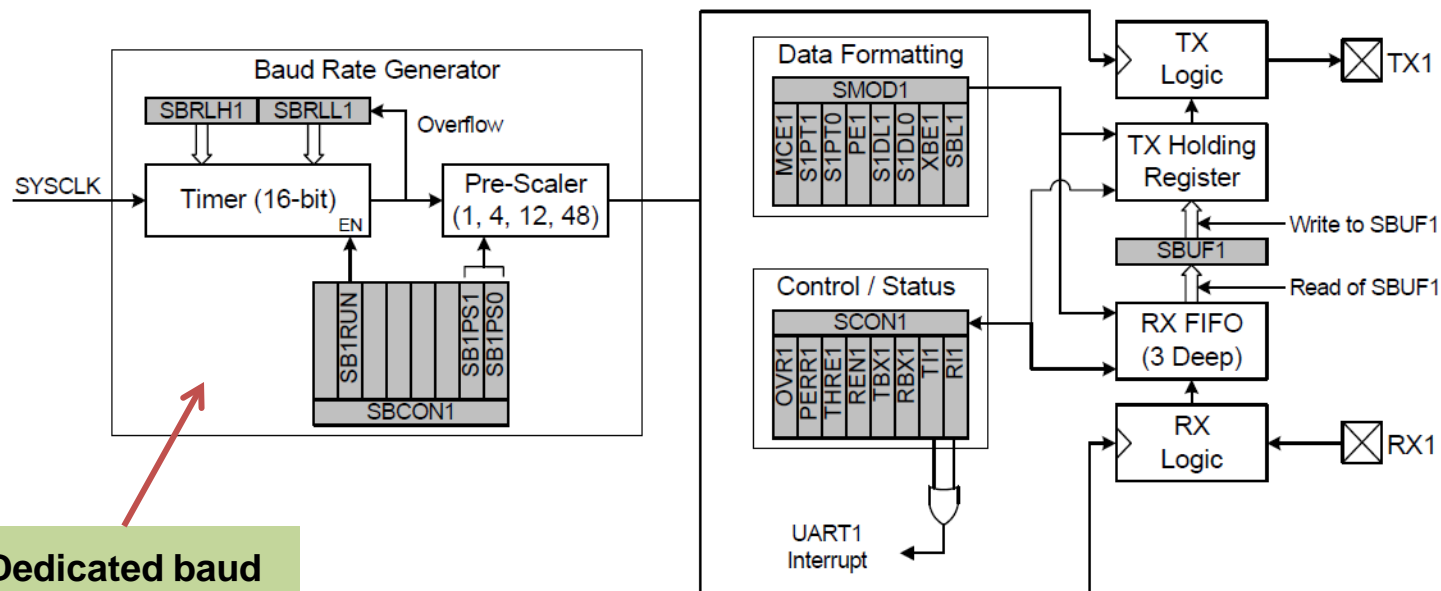
SILICON LABS

[www.silabs.com](http://www.silabs.com)

# 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**

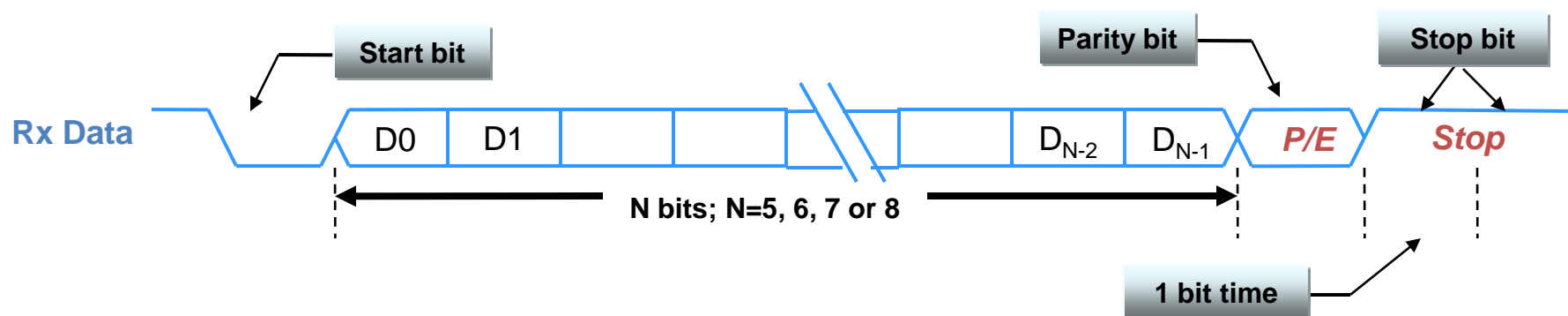


Dedicated baud rate generator

Enhanced UART Block Diagram

# Operating Modes

- 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 9<sup>TH</sup> bit for multi-processor communications
  - Parity can be enabled or disabled
  - Stop bit length can be changed



**P/E** Parity bit can be enabled or disabled or the bit time can be used for an extra bit  
**Stop** 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:

$$BaudRate = \frac{SYSCLK}{(65536 - (SBRLH1:SBRL1))} \times \frac{1}{2} \times \frac{1}{Prescaler}$$

## **Baud Rate Example:**

**Desired baud rate = 57600 baud**

**Clock input to Timer 1 = System clock = 48 MHz**

**Changing above equation:**

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

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

$$SBRL = 65120$$



SILICON LABS

[www.silabs.com](http://www.silabs.com)

# Silicon Labs Tools for USB Development

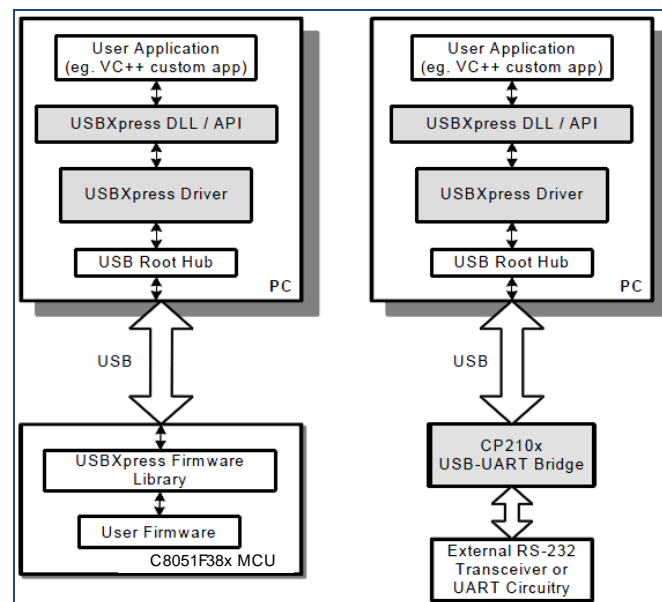
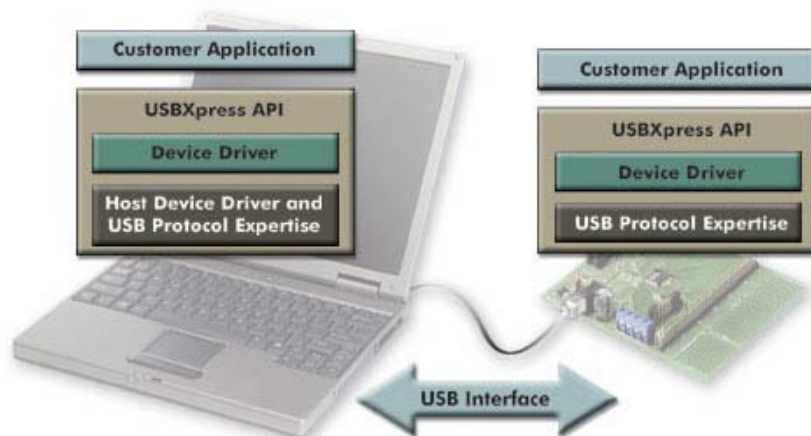


# Firmware Examples

- **Installed as part of the Silicon labs IDE and found at**
  - <http://www.silabs.com/PRODUCTS/MCU/Pages/SoftwareDownloads.aspx>
- **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



- **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



C8051F380DK Development Kit

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

## ➤ TOOLSTICK381DC

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



SILICON LABS

[www.silabs.com](http://www.silabs.com)

[www.silabs.com/USB](http://www.silabs.com/USB)