What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	AVR
Core Size	8-Bit
Speed	16MHz
Connectivity	I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, POR, PWM, WDT
Number of I/O	23
Program Memory Size	8KB (4K x 16)
Program Memory Type	FLASH
EEPROM Size	512 x 8
RAM Size	1K x 8
Voltage - Supply (Vcc/Vdd)	2.7V ~ 5.5V
Data Converters	A/D 8x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 105°C (TA)
Mounting Type	Surface Mount
Package / Case	32-TQFP
Supplier Device Package	32-TQFP (7x7)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/atmega8a-an

1. Description

The Atmel AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega8A provides the following features: 8K bytes of In-System Programmable Flash with Read-While- Write capabilities, 512 bytes of EEPROM, 1K byte of SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, one byte oriented Two-wire Serial Interface, a 6-channel ADC (eight channels in TQFP and QFN/MLF packages) with 10-bit accuracy, a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, and five software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, one SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next Interrupt or Hardware Reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low-power consumption.

Atmel offers the QTouch library for embedding capacitive touch buttons, sliders and wheels functionality into AVR microcontrollers. The patented charge-transfer signal acquisition offers robust sensing and includes fully debounced reporting of touch keys and includes Adjacent Key Suppression® (AKS®) technology for unambiguous detection of key events. The easy-to-use QTouch Composer allows you to explore, develop and debug your own touch applications.

The device is manufactured using Atmel's high density non-volatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core. The Boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega8A is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The device is supported with a full suite of program and system development tools including: C Compilers, Macro Assemblers, Program Debugger/Simulators, In-Circuit Emulators, and Evaluation kit.

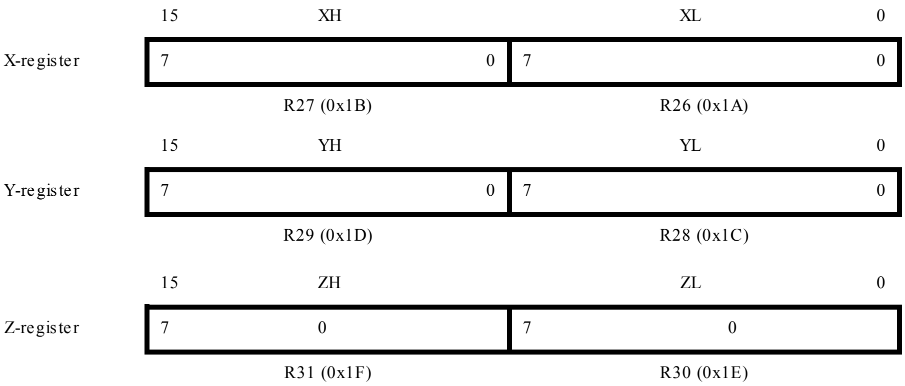
6. I/O Multiplexing

Each pin is by default controlled by the PORT as a general purpose I/O and alternatively it can be assigned to one of the peripheral functions. This table describes the peripheral signals multiplexed to the PORT I/O pins.

Table 6-1 PORT Function Multiplexing

PAD	Pin #	EXTINT	PCINT	AC	Custom	OSC	TC1(16-bit)	TC2(8-bit)	USART	SPI	Misc
PD[4]	14		PCINT20	ACO	-	-	O1CA	-		-	
PB[6]	1		PCINT06	-	-	EXTCLK	-	-	-	-	
PD[5]	2		PCINT21	AINP1	-	-	CLK1		-	-	SII
PD[6]	3		PCINT22	AINP0	-	-	ICP1	-	-	-	SDO
PD[7]	4		PCINT23	AINN0	-	-	-	TC2-OCB	-	-	SDI
PB[2]	5		PCINT02	-	CLO0	CLKOUT	TC1-OCB	-	-	SS	
PB[3]	6		PCINT03	-	-	-		TC2-OCA	TXD	MOSI	
PB[4]	7		PCINT04	-	-	-	-	-	RXD	MISO	
PB[5]	8		PCINT05	-	CLO1	-	-	-	XCK	SCK	
PC[4]	9		PCINT12	AINN1	-	-	-	-	-	-	
PC[5]	10	INT0	PCINT13	AINN2	-	-	-	-	-	-	
PC[6]/ RESET	13		PCINT14	-		-	-	-	-	-	HVRST/dW
VCC	11										
GND	12										

Figure 11-3 The X-, Y- and Z-Registers



In the different addressing modes these address registers have functions as fixed displacement, automatic increment, and automatic decrement (see the Instruction Set Reference for details).

11.5. Stack Pointer

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. Note that the Stack is implemented as growing from higher to lower memory locations. The Stack Pointer Register always points to the top of the Stack. The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. A Stack PUSH command will decrease the Stack Pointer.

The Stack in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. Initial Stack Pointer value equals the last address of the internal SRAM and the Stack Pointer must be set to point above start of the SRAM, see Figure *Data Memory Map* in *SRAM Data Memory*.

See table below for Stack Pointer details.

Table 11-1 Stack Pointer instructions

Instruction	Stack pointer	Description
PUSH	Decremented by 1	Data is pushed onto the stack
CALL ICALL RCALL	Decremented by 2	Return address is pushed onto the stack with a subroutine call or interrupt
POP	Incremented by 1	Data is popped from the stack
RET RETI	Incremented by 2	Return address is popped from the stack with return from subroutine or return from interrupt

The Atmel AVR Stack Pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used is implementation dependent. Note that the data space in some implementations of the AVR architecture is so small that only SPL is needed. In this case, the SPH Register will not be present.

Related Links

SRAM Data Memory on page 34

Table 13-4 Start-up Times for the Crystal Oscillator Clock Selection

CKSEL0	SUT1:0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
0	00	258 CK[(1)]	4.1ms	Ceramic resonator, fast rising power
0	01	258 CK[(1)]	65ms	Ceramic resonator, slowly rising power
0	10	1K CK[(2)]	–	Ceramic resonator, BOD enabled
0	11	1K CK[(2)]	4.1ms	Ceramic resonator, fast rising power
1	00	1K CK[(2)]	65ms	Ceramic resonator, slowly rising power
1	01	16K CK	–	Crystal Oscillator, BOD enabled
1	10	16K CK	4.1ms	Crystal Oscillator, fast rising power
1	11	16K CK	65ms	Crystal Oscillator, slowly rising power

Note:

1. These options should only be used when not operating close to the maximum frequency of the device, and only if frequency stability at start-up is not important for the application. These options are not suitable for crystals.
2. These options are intended for use with ceramic resonators and will ensure frequency stability at start-up. They can also be used with crystals when not operating close to the maximum frequency of the device, and if frequency stability at start-up is not important for the application.

13.4. Low-frequency Crystal Oscillator

To use a 32.768kHz watch crystal as the clock source for the device, the Low-frequency Crystal Oscillator must be selected by setting the CKSEL Fuses to “1001”. The crystal should be connected as shown in Figure 13-2 Crystal Oscillator Connections on page 46. By programming the CKOPT Fuse, the user can enable internal capacitors on XTAL1 and XTAL2, thereby removing the need for external capacitors. The internal capacitors have a nominal value of 36pF.

When this Oscillator is selected, start-up times are determined by the SUT Fuses as shown in the table below.

Table 13-5 Start-up Times for the Low-frequency Crystal Oscillator Clock Selection

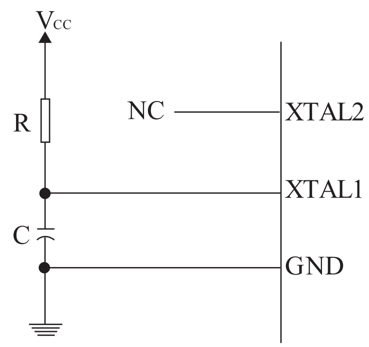
SUT1:0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	1K CK[(1)]	4.1ms	Fast rising power or BOD enabled
01	1K CK[(1)]	65ms	Slowly rising power
10	32K CK	65ms	Stable frequency at start-up
11	Reserved		

Note: 1. These options should only be used if frequency stability at start-up is not important for the application.

13.5. External RC Oscillator

For timing insensitive applications, the external RC configuration shown in the figure below can be used. The frequency is roughly estimated by the equation f = 1/(3RC). C should be at least 22pF. By programming the CKOPT Fuse, the user can enable an internal 36pF capacitor between XTAL1 and GND, thereby removing the need for an external capacitor.

Figure 13-3 External RC Configuration



The Oscillator can operate in four different modes, each optimized for a specific frequency range. The operating mode is selected by the fuses CKSEL3:0 as shown in the following table.

Table 13-6 External RC Oscillator Operating Modes

CKSEL3:0	Frequency Range (MHz)
0101	0.1 - 0.9
0110	0.9 - 3.0
0111	3.0 - 8.0
1000	8.0 - 12.0

When this Oscillator is selected, start-up times are determined by the SUT Fuses as shown in the table below.

Table 13-7 Start-up Times for the External RC Oscillator Clock Selection

SUT1:0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	18 CK	–	BOD enabled
01	18 CK	4.1ms	Fast rising power
10	18 CK	65ms	Slowly rising power
11	6 CK[1]	4.1ms	Fast rising power or BOD enabled

Note: 1. This option should not be used when operating close to the maximum frequency of the device.

13.6. Calibrated Internal RC Oscillator

The calibrated internal RC Oscillator provides a fixed 1.0, 2.0, 4.0, or 8.0MHz clock. All frequencies are nominal values at 5V and 25°C. This clock may be selected as the system clock by programming the CKSEL Fuses as shown in the next table. If selected, it will operate with no external components. The

22.11.5. TIMSK – Timer/Counter Interrupt Mask Register

When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these offset addresses. The device is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

Name: TIMSK
Offset: 0x39
Reset: 0x00
Property: When addressing I/O Registers as data space the offset address is 0x59

Bit	7	6	5	4	3	2	1	0
	OCIE2	TOIE2						
Access	R/W	R/W						
Reset	0	0						

Bit 7 – OCIE2: Timer/Counter2 Output Compare Match Interrupt Enable

When the OCIE2 bit is written to one and the I-bit in the Status Register is set (one), the Timer/Counter2 Compare Match interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter2 occurs (i.e., when the OCF2 bit is set in the Timer/Counter Interrupt Flag Register – TIFR).

Bit 6 – TOIE2: Timer/Counter2 Overflow Interrupt Enable

When the TOIE2 bit is written to one and the I-bit in the Status Register is set (one), the Timer/Counter2 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter2 occurs (i.e., when the TOV2 bit is set in the Timer/Counter Interrupt Flag Register – TIFR).

Note: 1. See *About Code Examples*.

The following code examples show how to initialize the SPI as a Slave and how to perform a simple reception.

Assembly Code Example[1]

```
SPI_SlaveInit:
   ; Set MISO output, all others input
   ldi     r17,(1<<DD_MISO)
   out     DDR_SPI,r17
   ; Enable SPI
   ldi     r17,(1<<SPE)
   out     SPCR,r17
   ret
SPI_SlaveReceive:
   ; Wait for reception complete
   sbis    SPSR,SPIF
   rjmp    SPI_SlaveReceive
   ; Read received data and return
   in      r16,SPDR
   ret
```

C Code Example[1]

```
void SPI_SlaveInit(void)
{
   /* Set MISO output, all others input */
   DDR_SPI = (1<<DD_MISO);
   /* Enable SPI */
   SPCR = (1<<SPE);
}
char SPI_SlaveReceive(void)
{
   /* Wait for reception complete */
   while(!(SPSR & (1<<SPIF)))
      ;
   /* Return Data Register */
   return SPDR;
}
```

Note: 1. See *About Code Examples*.

Related Links

Pin Configurations on page 13
Alternate Functions of Port B on page 83
Alternate Port Functions on page 81
About Code Examples on page 23

23.3. $\overline{\text{SS}}$ Pin Functionality

23.3.1. Slave Mode

When the SPI is configured as a Slave, the Slave Select ($\overline{\text{SS}}$) pin is always input. When $\overline{\text{SS}}$ is held low, the SPI is activated, and MISO becomes an output if configured so by the user. All other pins are inputs.

C Code Example[(1)]

```
#define FOSC 1843200 // Clock Speed
#define BAUD 9600
#define MYUBRR FOSC/16/BAUD-1
void main( void )
{
   ...
   USART_Init(MYUBRR)
   ...
}
void USART_Init( unsigned int ubrr)
{
   /*Set baud rate */
   UBRR0H = (unsigned char)(ubrr>>8);
   UBRR0L = (unsigned char)ubrr;
   Enable receiver and transmitter */
   UCSRB = (1<<RXEN)|(1<<TXEN);
   /* Set frame format: 8data, 2stop bit */
   UCSRC = (1<<URSEL)|(1<<USBS)|(3<<UCSZ0);
}
```

Note: 1. See *About Code Examples*.

More advanced initialization routines can be written to include frame format as parameters, disable interrupts, and so on. However, many applications use a fixed setting of the baud and control registers, and for these types of applications the initialization code can be placed directly in the main routine, or be combined with initialization code for other I/O modules.

Related Links

About Code Examples on page 23

24.6. Data Transmission – The USART Transmitter

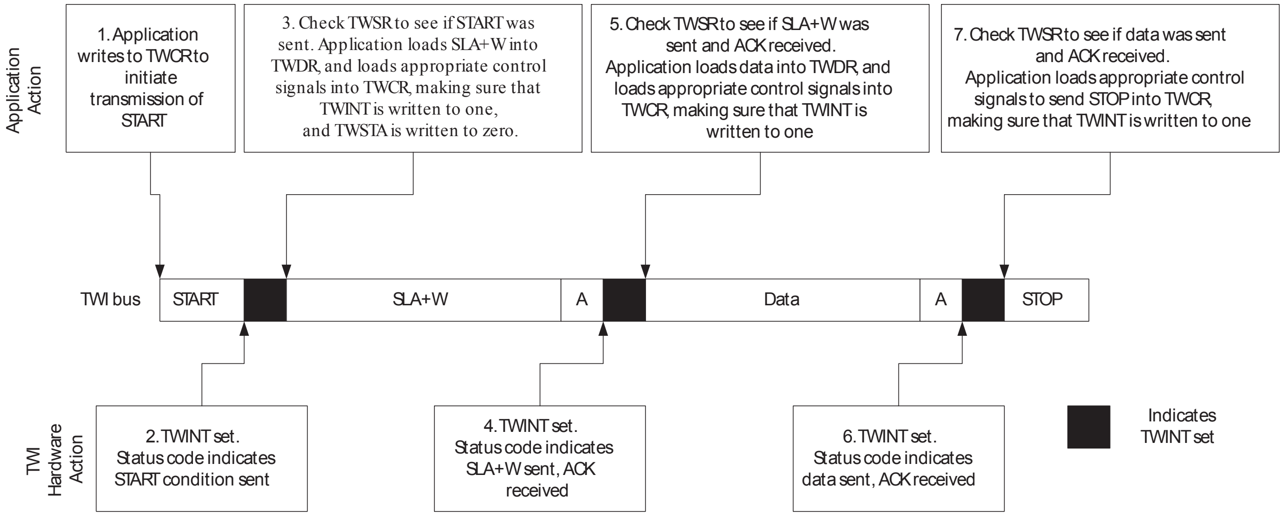
The USART Transmitter is enabled by setting the Transmit Enable (TXEN) bit in the UCSRB Register. When the Transmitter is enabled, the normal port operation of the TxD pin is overridden by the USART and given the function as the Transmitter's serial output. The baud rate, mode of operation and frame format must be set up once before doing any transmissions. If synchronous operation is used, the clock on the XCK pin will be overridden and used as transmission clock.

24.6.1. Sending Frames with 5 to 8 Data Bits

A data transmission is initiated by loading the transmit buffer with the data to be transmitted. The CPU can load the transmit buffer by writing to the UDR I/O location. The buffered data in the transmit buffer will be moved to the Shift Register when the Shift Register is ready to send a new frame. The Shift Register is loaded with new data if it is in idle state (no ongoing transmission) or immediately after the last stop bit of the previous frame is transmitted. When the Shift Register is loaded with new data, it will transfer one complete frame at the rate given by the Baud Register, U2X bit or by XCK depending on mode of operation.

The following code examples show a simple USART transmit function based on polling of the Data Register Empty (UDRE) Flag. When using frames with less than eight bits, the most significant bits written to the UDR are ignored. The USART has to be initialized before the function can be used. For the assembly code, the data to be sent is assumed to be stored in Register R16.

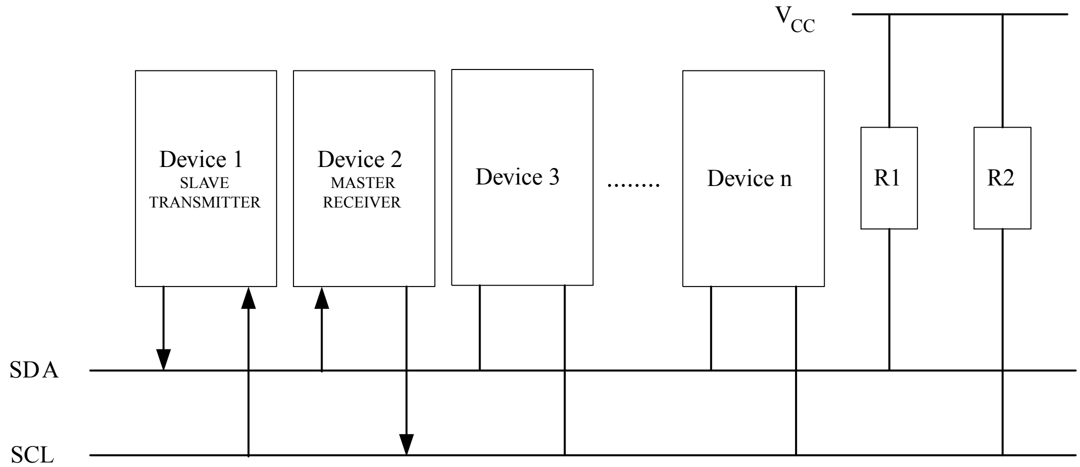
Figure 25-10 Interfacing the Application to the TWI in a Typical Transmission



1. The first step in a TWI transmission is to transmit a START condition. This is done by writing a specific value into TWCR, instructing the TWI hardware to transmit a START condition. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the START condition.
2. When the START condition has been transmitted, the TWINT Flag in TWCR is set, and TWSR is updated with a status code indicating that the START condition has successfully been sent.
3. The application software should now examine the value of TWSR, to make sure that the START condition was successfully transmitted. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load SLA+W into TWDR. Remember that TWDR is used both for address and data. After TWDR has been loaded with the desired SLA+W, a specific value must be written to TWCR, instructing the TWI hardware to transmit the SLA+W present in TWDR. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the address packet.
4. When the address packet has been transmitted, the TWINT Flag in TWCR is set, and TWSR is updated with a status code indicating that the address packet has successfully been sent. The status code will also reflect whether a Slave acknowledged the packet or not.
5. The application software should now examine the value of TWSR, to make sure that the address packet was successfully transmitted, and that the value of the ACK bit was as expected. If TWSR indicates otherwise, the application software might take some special action, like calling an error routine. Assuming that the status code is as expected, the application must load a data packet into TWDR. Subsequently, a specific value must be written to TWCR, instructing the TWI hardware to transmit the data packet present in TWDR. Which value to write is described later on. However, it is important that the TWINT bit is set in the value written. Writing a one to TWINT clears the flag. The TWI will not start any operation as long as the TWINT bit in TWCR is set. Immediately after the application has cleared TWINT, the TWI will initiate transmission of the data packet.
6. When the data packet has been transmitted, the TWINT Flag in TWCR is set, and TWSR is updated with a status code indicating that the data packet has successfully been sent. The status code will also reflect whether a Slave acknowledged the packet or not.

Status Code (TWSR) Prescaler Bits are 0	Status of the 2-wire Serial Bus and 2-wire Serial Interface Hardware	Application Software Response					Next Action Taken by TWI Hardware
		To/from TWDR	To TWCR				
			STA	STO	TWI NT	TWE A	
0x98	Previously addressed with general call; data has been received; NOT ACK has been returned	Read data byte or Read data byte or Read data byte or Read data byte	0 0 1 1	0 0 0 0	1 1 1 1	0 1 0 1	Switched to the not addressed Slave mode; no recognition of own SLA or GCA Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1” Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free
0xA0	A STOP condition or repeated START condition has been received while still addressed as Slave	No action	0 0 1 1	0 0 0 0	1 1 1 1	0 1 0 1	Switched to the not addressed Slave mode; no recognition of own SLA or GCA Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1” Switched to the not addressed Slave mode; no recognition of own SLA or GCA; a START condition will be transmitted when the bus becomes free Switched to the not addressed Slave mode; own SLA will be recognized; GCA will be recognized if TWGCE = “1”; a START condition will be transmitted when the bus becomes free

Figure 25-17 Data Transfer in Slave Transmitter Mode



To initiate the SR mode, the TWI (Slave) Address Register (TWAR) and the TWI Control Register (TWCR) must be initialized as follows:

The upper seven bits of TWAR are the address to which the 2-wire Serial Interface will respond when addressed by a Master (TWAR.TWA[6:0]). If the LSB of TWAR is written to TWAR.TWGCI=1, the TWI will respond to the general call address (0x00), otherwise it will ignore the general call address.

TWCR must hold a value of the type TWCR=0100010x - TWEN must be written to one to enable the TWI. The TWEA bit must be written to one to enable the acknowledgement of the device's own slave address or the general call address. TWSTA and TWSTO must be written to zero.

When TWAR and TWCR have been initialized, the TWI waits until it is addressed by its own slave address (or the general call address if enabled) followed by the data direction bit. If the direction bit is "1" (read), the TWI will operate in ST mode, otherwise SR mode is entered. After its own slave address and the write bit have been received, the TWINT Flag is set and a valid status code can be read from TWSR. The status code is used to determine the appropriate software action. The appropriate action to be taken for each status code is detailed in the table below. The ST mode may also be entered if arbitration is lost while the TWI is in the Master mode (see state 0xB0).

If the TWCR.TWEA bit is written to zero during a transfer, the TWI will transmit the last byte of the transfer. State 0xC0 or state 0xC8 will be entered, depending on whether the Master Receiver transmits a NACK or ACK after the final byte. The TWI is switched to the not addressed Slave mode, and will ignore the Master if it continues the transfer. Thus the Master Receiver receives all '1' as serial data. State 0xC8 is entered if the Master demands additional data bytes (by transmitting ACK), even though the Slave has transmitted the last byte (TWEA zero and expecting NACK from the Master).

While TWCR.TWEA is zero, the TWI does not respond to its own slave address. However, the 2-wire Serial Bus is still monitored and address recognition may resume at any time by setting TWEA. This implies that the TWEA bit may be used to temporarily isolate the TWI from the 2-wire Serial Bus.

In all sleep modes other than Idle mode, the clock system to the TWI is turned off. If the TWEA bit is set, the interface can still acknowledge its own slave address or the general call address by using the 2-wire Serial Bus clock as a clock source. The part will then wake up from sleep and the TWI will hold the SCL clock will low during the wake up and until the TWINT Flag is cleared (by writing '1' to it). Further data transmission will be carried out as normal, with the AVR clocks running as normal. Observe that if the AVR is set up with a long start-up time, the SCL line may be held low for a long time, blocking other data transmissions.

Note: The 2-wire Serial Interface Data Register (TWDR) does not reflect the last byte present on the bus when waking up from these Sleep modes.

25.8.3. TWSR – TWI Status Register

When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these offset addresses. The device is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

Name: TWSR
Offset: 0x01
Reset: 0xF8
Property: When addressing I/O Registers as data space the offset address is 0x21

Bit	7	6	5	4	3	2	1	0
	TWS4	TWS3	TWS2	TWS1	TWS0		TWPS1	TWPS0
Access	R	R	R	R	R		R/W	R/W
Reset	0	0	0	0	1		0	0

Bits 7:3 – TWSn: TWI Status Bit 7 [n = 7:3]
The TWS[7:3] reflect the status of the TWI logic and the 2-wire Serial Bus. The different status codes are described later in this section. Note that the value read from TWSR contains both the 5-bit status value and the 2-bit prescaler value. The application designer should mask the prescaler bits to zero when checking the Status bits. This makes status checking independent of prescaler setting. This approach is used in this datasheet, unless otherwise noted.

Bits 1:0 – TWPSn: TWI Prescaler [n = 1:0]
These bits can be read and written, and control the bit rate prescaler.

Table 25-8 TWI Bit Rate Prescaler

TWPS1	TWPS0	Prescaler Value
0	0	1
0	1	4
1	0	16
1	1	64

To calculate bit rates, refer to Bit Rate Generator Unit. The value of TWPS1:0 is used in the equation.

25.8.4. TWDR – TWI Data Register

When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these offset addresses. The device is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

In Transmit mode, TWDR contains the next byte to be transmitted. In Receive mode, the TWDR contains the last byte received. It is writable while the TWI is not in the process of shifting a byte. This occurs when the TWI Interrupt Flag (TWINT) is set by hardware. Note that the Data Register cannot be initialized by the user before the first interrupt occurs. The data in TWDR remains stable as long as TWINT is set. While data is shifted out, data on the bus is simultaneously shifted in. TWDR always contains the last byte present on the bus, except after a wake up from a sleep mode by the TWI interrupt. In this case, the contents of TWDR is undefined. In the case of a lost bus arbitration, no data is lost in the transition from Master to Slave. Handling of the ACK bit is controlled automatically by the TWI logic, the CPU cannot access the ACK bit directly.

Name: TWDR
Offset: 0x03
Reset: 0xFF
Property: When addressing I/O Registers as data space the offset address is 0x23

Bit	7	6	5	4	3	2	1	0
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	1

Bits 7:0 – TWDn: TWI Data [n = 7:0]
These eight bits constitute the next data byte to be transmitted, or the latest data byte received on the 2-wire Serial Bus.

28. Boot Loader Support – Read-While-Write Self-Programming

28.1. Features

- Read-While-Write Self-Programming
- Flexible Boot Memory Size
- High Security (Separate Boot Lock Bits for a Flexible Protection)
- Separate Fuse to Select Reset Vector
- Optimized Page[(1)] Size
- Code Efficient Algorithm
- Efficient Read-Modify-Write Support

Note: 1. A page is a section in the Flash consisting of several bytes (see Table. No. of Words in a Page and No. of Pages in the Flash in *Page Size*) used during programming. The page organization does not affect normal operation.

Related Links

Page Size on page 286

28.2. Overview

In this device, the Boot Loader Support provides a real Read-While-Write Self-Programming mechanism for downloading and uploading program code by the MCU itself. This feature allows flexible application software updates controlled by the MCU using a Flash-resident Boot Loader program. The Boot Loader program can use any available data interface and associated protocol to read code and write (program) that code into the Flash memory, or read the code from the program memory. The program code within the Boot Loader section has the capability to write into the entire Flash, including the Boot Loader memory. The Boot Loader can thus even modify itself, and it can also erase itself from the code if the feature is not needed anymore. The size of the Boot Loader memory is configurable with fuses and the Boot Loader has two separate sets of Boot Lock bits which can be set independently. This gives the user a unique flexibility to select different levels of protection.

28.3. Application and Boot Loader Flash Sections

The Flash memory is organized in two main sections, the Application section and the Boot Loader section. The size of the different sections is configured by the BOOTSZ Fuses. These two sections can have different level of protection since they have different sets of Lock bits.

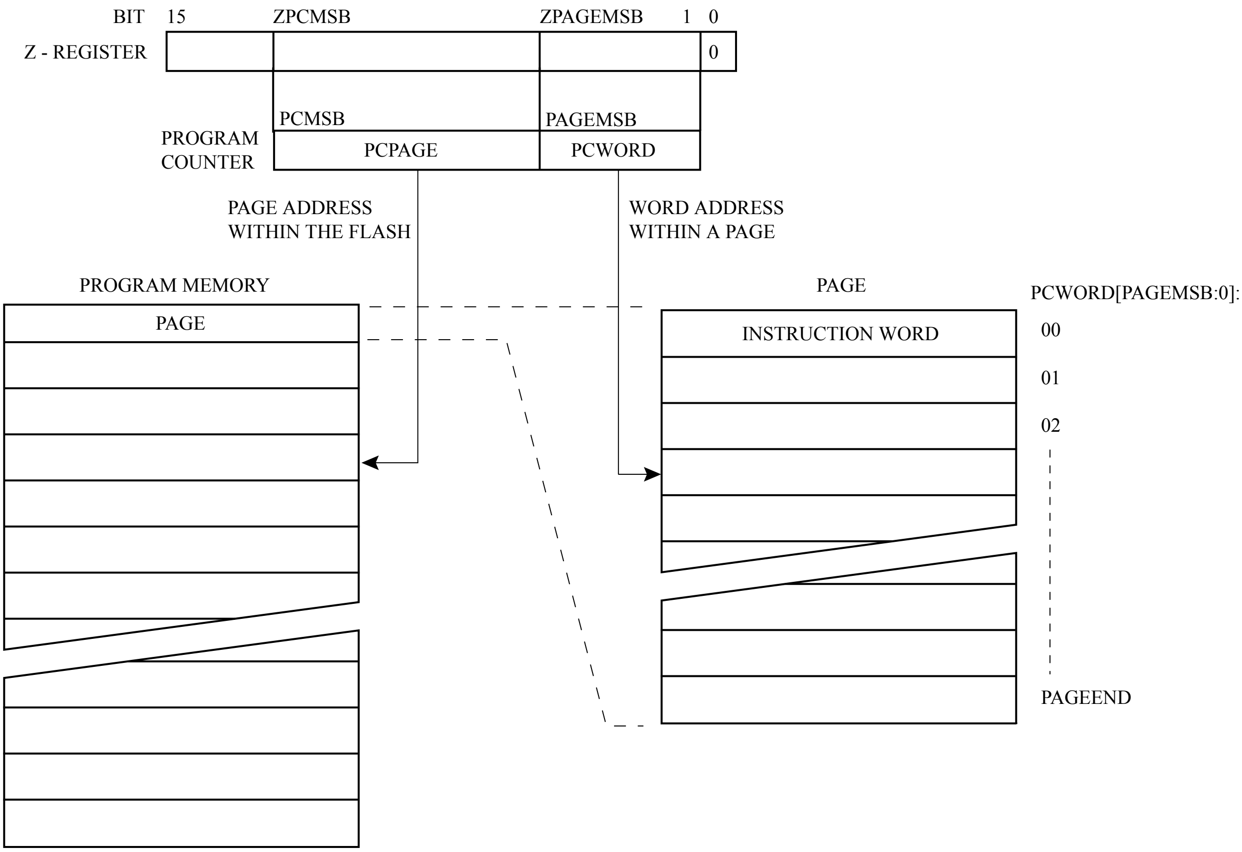
28.3.1. Application Section

The Application section is the section of the Flash that is used for storing the application code. The protection level for the Application section can be selected by the application Boot Lock bits (Boot Lock bits 0). The Application section can never store any Boot Loader code since the SPM instruction is disabled when executed from the Application section.

28.3.2. BLS – Boot Loader Section

While the Application section is used for storing the application code, the The Boot Loader software must be located in the BLS since the SPM instruction can initiate a programming when executing from the BLS only. The SPM instruction can access the entire Flash, including the BLS itself. The protection level for the Boot Loader section can be selected by the Boot Loader Lock bits (Boot Lock bits 1).

Figure 28-3 Addressing the Flash During SPM(1)



Note:

1. The different variables used in the figure are listed in Table 28-8 Explanation of Different Variables used in Figure and the Mapping to the Z-pointer, ATmega8A on page 279.
2. PCPAGE and PCWORD are listed in table *Number of Words in a Page and number of Pages in the Flash* in the *Signal Names* section.

28.8. Self-Programming the Flash

The program memory is updated in a page by page fashion. Before programming a page with the data stored in the temporary page buffer, the page must be erased. The temporary page buffer is filled one word at a time using SPM and the buffer can be filled either before the Page Erase command or between a Page Erase and a Page Write operation:

Alternative 1, fill the buffer before a Page Erase

- Fill temporary page buffer
- Perform a Page Erase
- Perform a Page Write

Alternative 2, fill the buffer after Page Erase

- Perform a Page Erase
- Fill temporary page buffer
- Perform a Page Write

High Fuse Byte	Bit No.	Description	Default Value
BOOTSZ1	2	Select Boot Size (see *ATmega8A Boot Loader Parameters*)	0 (programmed)[3]
BOOTSZ0	1	Select Boot Size (see *ATmega8A Boot Loader Parameters*)	0 (programmed)[3]
BOOTRST	0	Select Reset Vector	1 (unprogrammed)

Note:

1. The SPIEN Fuse is not accessible in Serial Programming mode.
2. The CKOPT Fuse functionality depends on the setting of the CKSEL bits, see *Clock Sources* for details.
3. The default value of BOOTSZ1:0 results in maximum Boot Size. See *Boot Loader Parameters* for details.
4. When programming the RSTDISBL Fuse Parallel Programming has to be used to change fuses or perform further programming.

Table 29-4 Fuse Low Byte

Low Fuse Byte	Bit No.	Description	Default Value
BODLEVEL	7	Brown out detector trigger level	1 (unprogrammed)
BODEN	6	Brown out detector enable	1 (unprogrammed, BOD disabled)
SUT1	5	Select start-up time	1 (unprogrammed)[1]
SUT0	4	Select start-up time	0 (programmed)[1]
CKSEL3	3	Select Clock source	0 (programmed)[2]
CKSEL2	2	Select Clock source	0 (programmed)[2]
CKSEL1	1	Select Clock source	0 (programmed)[2]
CKSEL0	0	Select Clock source	1 (unprogrammed)[2]

Note:

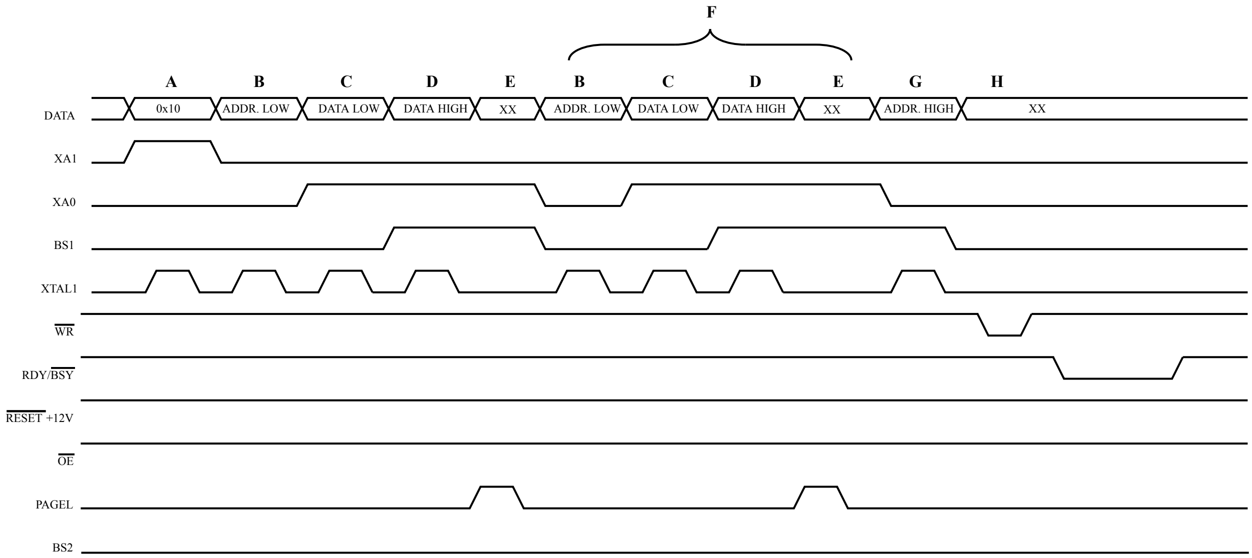
1. The default value of SUT1:0 results in maximum start-up time. See table *Start-up Times for the Internal Calibrated RC Oscillator Clock Selection* in *Calibrated Internal RC Oscillator* of the System Clock and Clock Options chapter for details.
2. The default setting of CKSEL3:0 results in internal RC Oscillator @ 1MHz. See table *Internal Calibrated RC Oscillator Operating Modes* in *Calibrated Internal RC Oscillator* of the System Clock and Clock Options chapter for details.

The status of the Fuse bits is not affected by Chip Erase. Note that the Fuse bits are locked if Lock bit1 (LB1) is programmed. Program the Fuse bits before programming the Lock bits.

Related Links

Clock Sources on page 45
ATmega8A Boot Loader Parameters on page 278
Calibrated Internal RC Oscillator on page 48

Figure 29-3 Programming the Flash Waveform



Note: "XX" is don't care. The letters refer to the programming description above.

29.7.5. Programming the EEPROM

The EEPROM is organized in pages, see Page Size on page 286, Table 29-7 Number of Words in a Page and Number of Pages in the EEPROM on page 286. When programming the EEPROM, the program data is latched into a page buffer. This allows one page of data to be programmed simultaneously. The programming algorithm for the EEPROM data memory is as follows (For details on Command, Address and Data loading, refer to Programming the Flash on page 289):

1. Step A: Load Command "0001 0001".
2. Step G: Load Address High Byte (0x00 - 0xFF).
3. Step B: Load Address Low Byte (0x00 - 0xFF).
4. Step C: Load Data (0x00 - 0xFF).
5. Step E: Latch data (give PAGEL a positive pulse).
6. Step K:Repeat 3 through 5 until the entire buffer is filled.
7. Step L: Program EEPROM page
   - 7.1. Set BS1 to "0".
   - 7.2. Give $\overline{WR}$ a negative pulse. This starts programming of the EEPROM page. RDY/$\overline{BSY}$ goes low.
   - 7.3. Wait until to RDY/$\overline{BSY}$ goes high before programming the next page. Refer to the figure below for signal waveforms.

Figure 32-2 Active Supply Current vs. Frequency (1 - 16MHz)

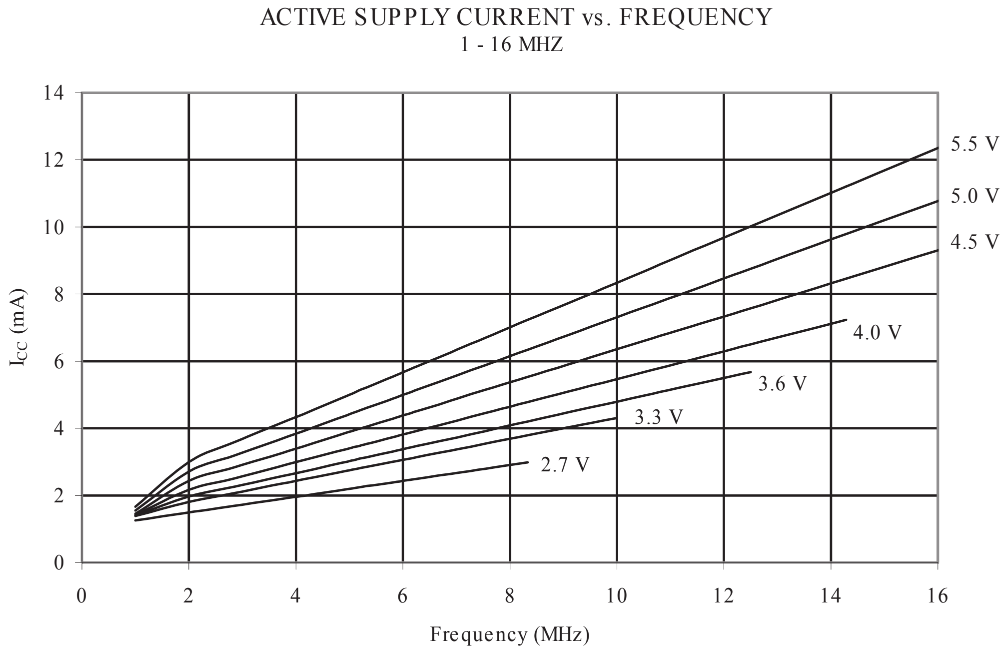


Figure 32-3 Active Supply Current vs. V_{CC} (Internal RC Oscillator, 8MHz)

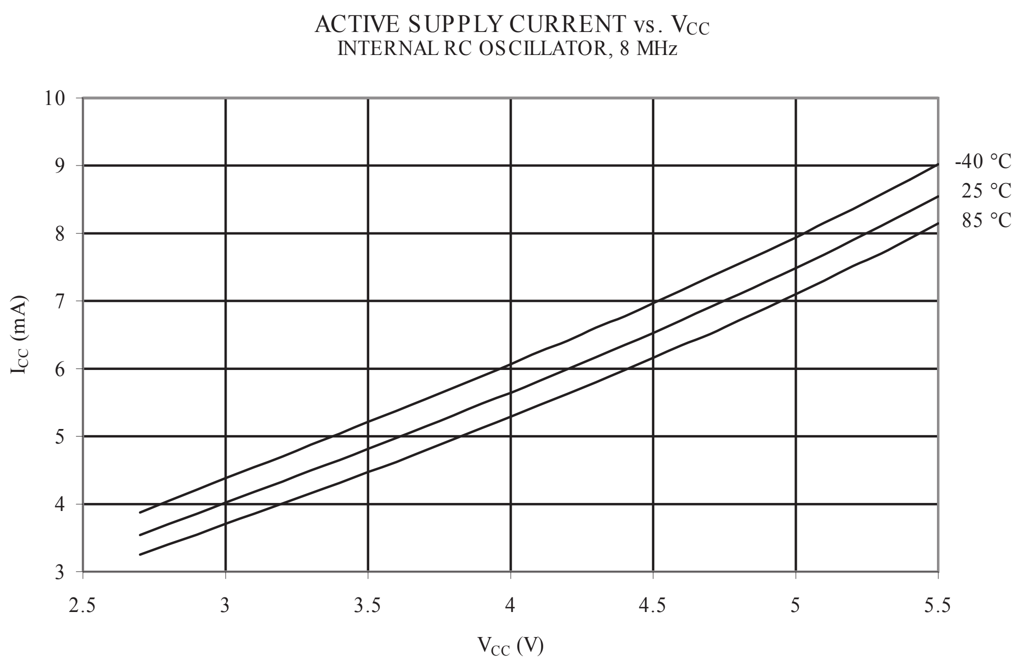


Figure 32-47 Bandgap Voltage vs. V_{CC}

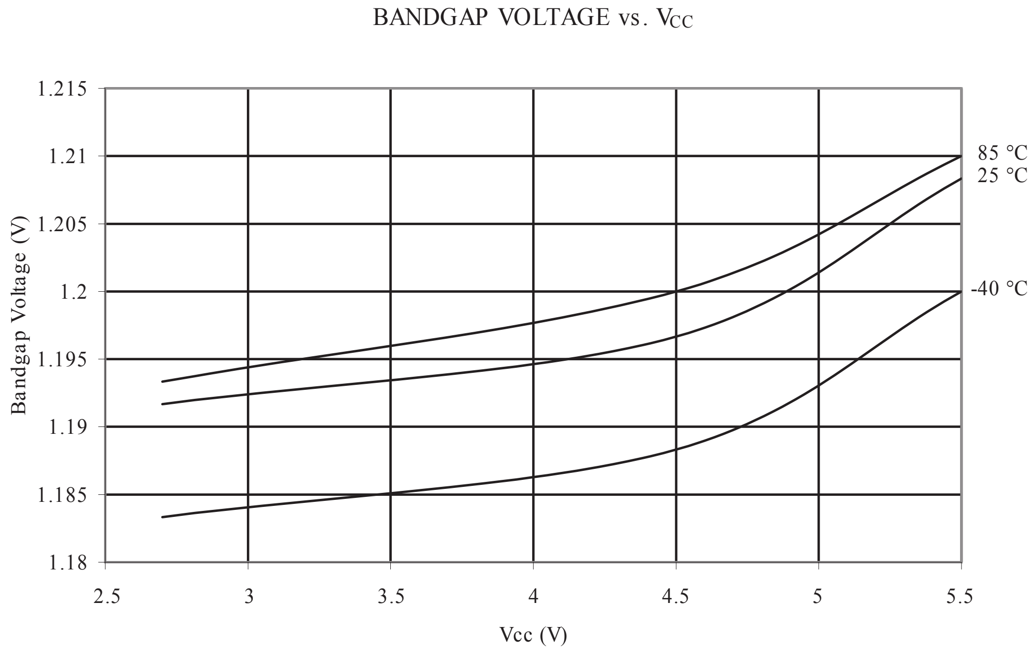


Figure 32-48 Analog Comparator Offset Voltage vs. Common Mode Voltage (V_{CC} = 5V)

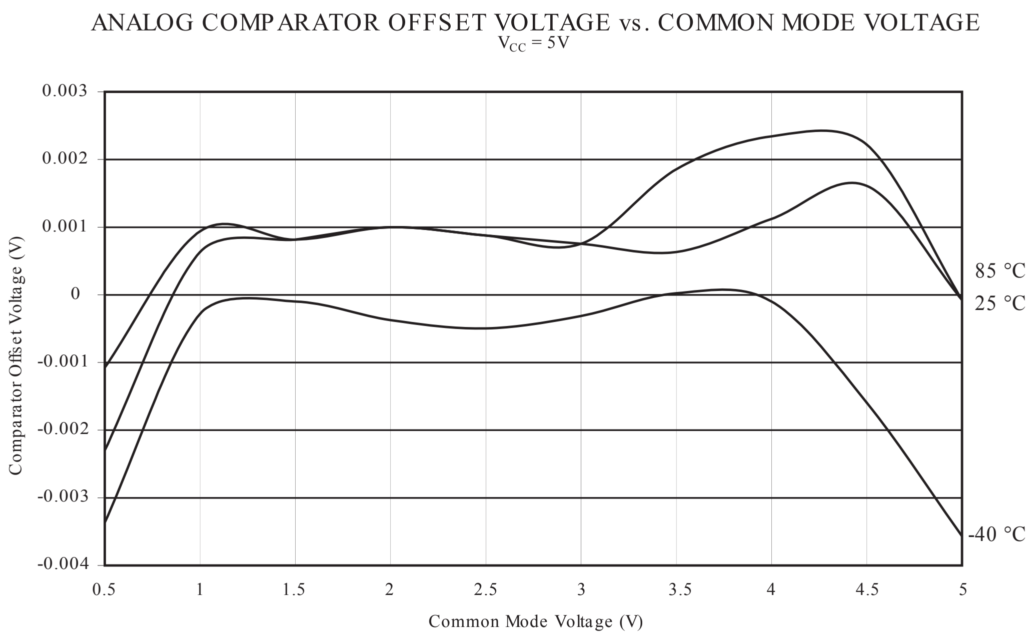


Figure 33-41 Calibrated 4MHz RC Oscillator vs. V_{CC}

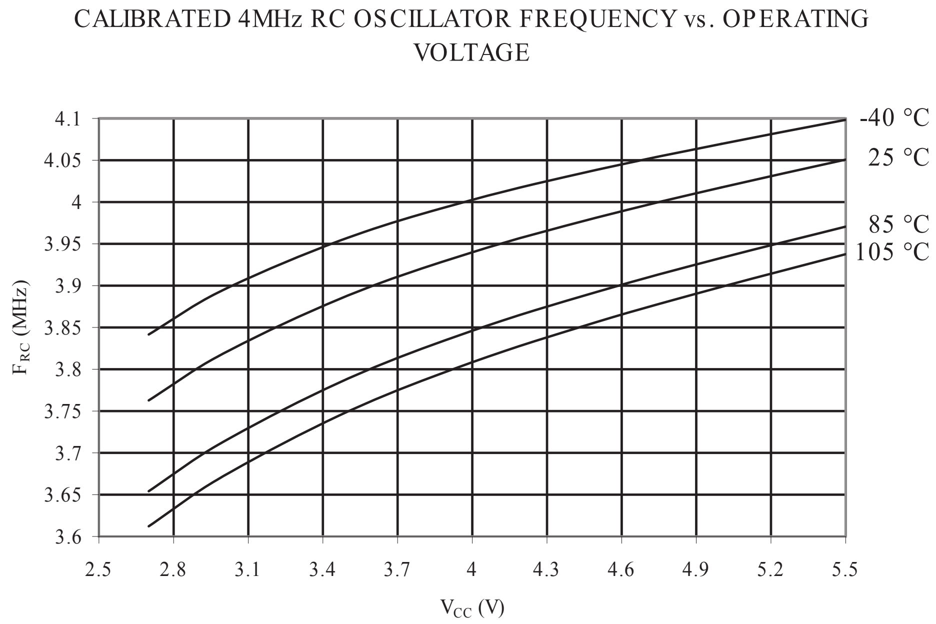


Figure 33-42 Calibrated 4MHz RC Oscillator vs. OSCCAL Value