AVR910: In-System Programming

Features
- **Complete In-System Programming Solution for AVR Microcontrollers**
- **Covers All AVR Microcontrollers with In-System Programming Support**
- **Reprogram Both Data Flash and Parameter EEPROM Memories**
- **Complete Schematics for Low-cost In-System Programmer**
- **Simple Three-wire SPI Programming Interface**

Introduction

In-System Programming allows programming and reprogramming of any AVR micro-controller positioned inside the end system. Using a simple Three-wire SPI interface, the In-System Programmer communicates serially with the AVR microcontroller, reprogramming all non-volatile memories on the chip.

In-System Programming eliminates the physical removal of chips from the system. This will save time, and money, both during development in the lab, and when updating the software or parameters in the field.

This application note shows how to design the system to support In-System Programming. It also shows how a low-cost In-System Programmer can be made, that will allow the target AVR microcontroller to be programmed from any PC equipped with a regular 9-pin serial port. Alternatively, the entire In-System Programmer can be built into the system allowing it to reprogram itself.

The Programming Interface

For In-System Programming, the programmer is connected to the target using as few wires as possible. To program any AVR microcontroller in any target system, a simple Six-wire interface is used to connect the programmer to the target PCB. Figure 1 below shows the connections needed.

The Serial Peripheral Interface (SPI) consists of three wires: Serial ClocK (SCK), Master In – Slave Out (MISO) and Master Out – Slave In (MOSI). When programming the AVR, the In-System Programmer always operate as the Master, and the target system always operate as the Slave.

The In-System Programmer (Master) provides the clock for the communication on the SCK Line. Each pulse on the SCK Line transfers one bit from the Programmer (Master) to the Target (Slave) on the Master Out – Slave In (MOSI) line. Simultaneously, each pulse on the SCK Line transfers one bit from the target (Slave) to the Programmer (Master) on the Master In – Slave Out (MISO) line.
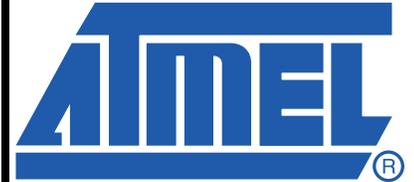
**Figure 1.** Six-wire Connection Between Programmer and Target System
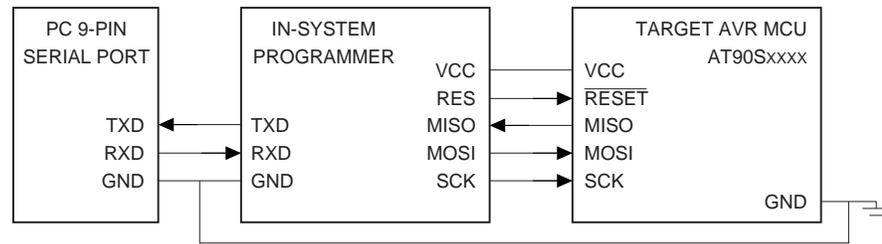
To assure proper communication on the three SPI lines, it is necessary to connect ground on the programmer to ground on the target (GND).

To enter and stay in Serial Programming mode, the AVR microcontroller reset line has to be kept active (low). Also, to perform a Chip Erase, the Reset has to be pulsed to end the Chip Erase cycle. To ease the programming task, it is preferred to let the programmer take control of the target microcontroller reset line to automate this process using a fourth control line (Reset).

To allow programming of targets running at any allowed voltage (2.7 - 6.0 V), the programmer can draw power from the target system (V_{CC}). This eliminate the need for a separate power supply for the programmer. Alternatively, the target system can be supplied from the programmer at programming time, eliminating the need to power the target system through its regular power connector for the duration of the programming cycle.

Figure 2 shows the connector used by this In-System Programmer to connect to the target system. The standard connector supplied is a 2 x 3 pin header contact, with pin spacing of 100 mils.

Figure 2. Recommended In-System Programming Interface Connector Layout (Top View)
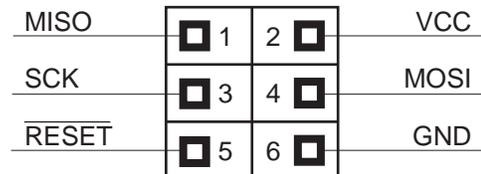
Hardware Design Considerations

To allow In-System Programming of the AVR microcontroller, the In-System Programmer must be able to override the pin functionality during programming. This section describes the details of each pin used for the programming operation.

GND

The In-System Programmer and target system need to operate with the same reference voltage. This is done by connecting ground of the target to ground of the programmer. No special considerations apply to this pin.

RESET

The target AVR microcontroller will enter Serial Programming mode only when its reset line is active (low). When erasing the chip, the reset line has to be toggled to end the erase cycle. To simplify this operation, it is recommended that the target reset can be controlled by the In-System Programmer.

Immediately after Reset has gone active, the In-System Programmer will start to communicate on the three dedicated SPI wires SCK, MISO, and MOSI. To avoid driver contention, a series resistor should be placed on each of the three dedicated lines if there is a possibility that external circuitry could be driving these lines. The connection is shown in Figure 3. The value of the resistors should be chosen depending on the circuitry connected to the SPI bus. Note that the

AVR microcontroller will automatically set all its I/O pins to inputs, with pull ups disabled, when Reset is active.

Table 1. Connections Required for In-System Programming

Pin	Name	Comment
SCK	Serial Clock	Programming clock, generated by the In-System Programmer (Master)
MOSI	Master Out – Slave In	Communication line from In-System Programmer (Master) to target AVR being programmed (Slave)
MISO	Master In – Slave Out	Communication line from target AVR (Slave) to In-System Programmer (Master)
GND	Common Ground	The two systems must share the same common ground
RESET	Target AVR MCU Reset	To enable In-System Programming, the target AVR Reset must be kept active. To simplify this, the In-System Programmer should control the target AVR Reset
V_{CC}	Target Power	To allow simple programming of targets operating at any voltage, the In-System Programmer can draw power from the target. Alternatively, the target can have power supplied through the In-System Programming connector for the duration of the programming cycle

Figure 3. Connecting ISP Programming Cable to Target SPI Bus
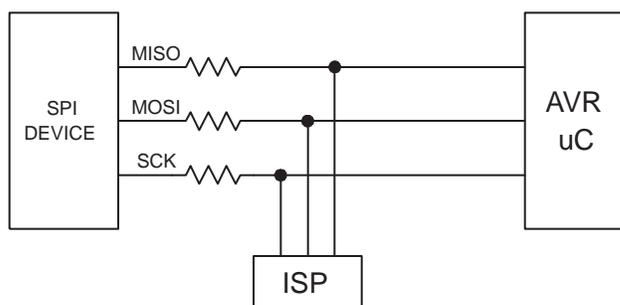
To avoid problems, the In-System Programmer should be able to keep the entire Target System Reset for the duration of the programming cycle. The target system should never attempt to drive the three SPI lines while Reset is active.

SCK

When programming the AVR in Serial mode, the In-System Programmer supplies clock information on the SCK pin. This pin is always driven by the programmer, and the target system should never attempt to drive this wire when target reset is active. Immediately after the Reset goes active, this pin will be driven to zero by the programmer. During this first phase of the programming cycle, keeping the SCK Line free from pulses is critical, as pulses will cause the target AVR to loose synchronization with the programmer. When in synchronization, the second byte ($53), will echo back when issuing the third byte of the programming enable instruction. If the $53 did not echo back, give Reset a positive pulse, and issue a new Programming Enable command. Note that all four bytes of the of the Programming Enable command must be sent before starting a new transmission.

The target AVR microcontroller will always set up its SCK pin to be an input with no pull up whenever Reset is active. See also the description of the Reset wire.

The minimum low and high periods for the Serial Clock (SCK) input are defined in the programming section of the datasheet. For the AT90S1200 they are defined as follows:

Low: >1 XTAL1 clock cycle

High: >4 XTAL1 clock cycles

MOSI
When programming the AVR in Serial mode, the In-System Programmer supplies data to the target on the MOSI pin. This pin is always driven by the programmer, and the target system should never attempt to drive this wire when target reset is active.

The target AVR microcontroller will always set up its MOSI pin to be an input with no pull up whenever Reset is active. See also the description of the Reset wire.

MISO
When Reset is applied to the target AVR microcontroller, the MISO pin is set up to be an input with no pull up. Only after the "Programming Enable" command has been correctly transmitted to the target will the target AVR microcontroller set its MISO pin to become an output. During this first time, the In-System programmer will apply its pull up to keep the MISO line stable until it is driven by the target microcontroller.

V$_{CC}$
When programming the target microcontroller, the programmer outputs need to stay within the ranges specified in the DC Characteristics.

To easily adapt to any target voltage, the programmer can draw all power required from the target system. This is allowed as the In-System Programmer will draw very little power from the target system, typically no more than 20 mA. The programmer shown in this application note operates in this mode.

As an alternative, the target system can have its power supplied from the programmer through the same connector used for the communication. This would allow the target to be programmed without applying power to the target externally.

Table 2. Recommendations when Designing Hardware Supporting In-System Programming

Pin	Recommendation
GND	Connect ground of the target to ground of the In-System Programmer
RESET	Allow the In-System Programmer to Reset the target system
SCK	When the target AVR microcontroller reset is active, this line should be controlled by the ISP Programmer. **Edges on this line after Reset is pulled low will be critical, and cause the target AVR microcontroller to loose synchronization with the programmer.** When programming, oscillations on this pin should be tolerated by the surrounding system when the AVR Reset is active
MOSI	When the target AVR microcontroller Reset is active, this line should be controlled by the ISP Programmer. When programming, oscillations on this pin should be tolerated by the surrounding system when the AVR Reset is active.
MISO	When the target AVR microcontroller Reset is active, this line should be allowed to become an output. When programming, oscillations on this pin should be tolerated by the surrounding system when the AVR Reset is active
V$_{CC}$	Allow the In-System Programmer to draw power from the target system, to adapt to any allowed target voltage. The maximum current needed to power the programmer will vary depending on the programmer being used.

Programming Protocol

Immediately after Reset goes active on the target AVR microcontroller, the chip is ready to enter Programming mode. The internal Serial Peripheral Interface (SPI) is activated, and is ready to accept instructions from the programmer. On the AT90S1200, it is very important to keep the

SCK pin stable, as one single edge will cause the target to loose synchronization with the programmer. For other devices use the synchronization algorithm specified in the datasheet. After pulling Reset low, wait at least 20 ms before issuing the first command.

Command Format

All commands have a common format consisting of four bytes. The first byte contains the command code, selecting operation and target memory. The second and third byte contain the address of the selected memory area. The fourth byte contains the data, going in either direction.

The data returned from the target is usually the data sent in the previous byte. Table 3 shows an example, where two consecutive commands are sent to the target. Notice how all bytes returned equal the bytes just received. Some commands return one byte from the target's memory. This byte is always returned in the last byte (byte 4). Data is alwa'ys sent on MOSI and MISO lines with most significant bit (MSB) first.

For details on available instructions, please refer to the Serial Programming section of the datasheet.

Enable Memory Access

When the Reset pin is first pulled active, the only instruction accepted by the SPI interface is "Programming Enable". Only this command will open for access to the Flash and EEPROM memories, and without this access, any other command issued will be ignored. Table 3 shows an example where memory access is enabled in the first command sent to the chip.

Table 3. Example, Enabling Memory Access and Erasing the Chip

Action	MOSI, Sent to Target AVR	MISO, Returned from Target AVR
Programming Enable	$AC 53 xx yy	$zz AC 53 xx
Read Device Code $1E at Address $00	$30 nn 00 mm	$yy 30 nn **1E**

After a "Programming Enable" command has been sent to the target, access is given to the non-volatile memories of the chip according to the current setting of the protecting Lock bits.

The target AVR microcontroller will not respond with an acknowledge to the "Programming Enable" command. To check if the command has been accepted by the target AVR microcontroller, the device code could be read. The device code is also known as the signature bytes.

Device Code

After the "Programming Enable" command has been successfully read by the SPI interface, the programmer can read the device code. The device code will identify the chip vendor (Atmel), the part family (AVR), Flash size in kilobytes, and family member (i.e., AT90S1200). The "Read Device Code" command format is found in the Serial Programming section of the datasheet. As an example, this command will, for the AT90S1200, be [$30, $XX, $adr, $code]. Valid addresses are $00, $01 and $02. Table 4 shows what the expected result will be.

Table 5 indicates that Device Code will sometimes read as $FF. If this happens, the part device code has not been programmed into the device. This does not indicate an error, but the part has to be manually identified to the programmer.

Device code $FF might also occur if there is no target ready or if the MISO line is constantly pulled high. The programmer can detect this situation by detecting that also a command sent to the target is returned as $FF.

If the target reports Vendor Code $00, Part Family $01, and Part Number $02, both Lock bits have been set. This prevents the memory blocks from responding, and the valued returned will be the byte just received from the programmer, which just happens to be the current address. To erase the Lock bits, it is necessary to perform a valid "Chip Erase".

Table 6 shows an example reading the Device Code from an AT90S1200.

Table 4. Allowed Device Codes

Address	Code	Valid Codes
$00	Vendor Code	$1E indicates manufactured by Atmel $00 indicates the device is locked, see below
$01	Part Family and Flash Size	$9n indicates AVR with 2^n kB Flash memory
$02	Part Number	Identifies the part, see the file avr910.asm for a complete listing of supported devices

Table 5. Part Number Identification Examples

Part Family and Flash Size	Part Number	Part
$90	$01	AT90S1200
$91	$01	AT90S2313
$92	$01	AT90S4414
$93	$01	AT90S8515
$FF	$FF	Device Code Erased (or Target Missing)
$01	$02	Device Locked

Table 6. Example, Reading the Device Code From an AT90S1200, Code $1E 90 01 Expected

Action	MOSI, Sent to Target AVR	MISO, Returned from Target AVR
Read Vendor Code at Address $00	$30 xx 00 yy	$zz 30 xx **1E**
Read Part Family and Memory Size at $01	$30 nn 01 mm	$yy 30 nn **90**
Read Part Number at Address $02	$30 xx 02 yy	$mm 30 xx **01**

Flash Program Memory Access

When the part has been identified, it is time to start accessing the Flash memory. A Chip Erase should be performed before programming the Flash memory. Depending on the target device the Flash is programmed using "Byte" or "Page" mode.

For devices with Byte Programming mode each Flash location is dressed and programmed individually. In Page Programming mode, a temporary Page buffer is first filled, and then programmed in one single write cycle. This mode reduces the total Flash programming time. A device will only have one of these modes available. A device with Byte Programming mode do not have the Page Programming option. A device with Page Programming mode of the Flash will, however, use byte programming for the EEPROM memory.

Regardless if the device uses Byte Programming mode or Page Programming mode the Flash will be read one byte at the time using the "Read Flash Program Memory" command. The command sends a memory address ($**aa bb**) to select a 16-bit word, and selects low or high byte with the **H** bit in the command byte (0 is low, 1 is high byte). The byte stored at this address is then returned from the target AVR microcontroller in byte 4.

Usually, each 16-bit word in Flash contains one AVR instruction. Assuming the instruction stored at address $104 is "add r16,r17", the op-code for this instruction would be stored as $0F01. Reading address $104 serially, the expected result returned in byte 4 will be $0F from the high

byte, and $01 from the low byte. The data on the MISO and MOSI lines will look like shown in Table 7.

Writing to the Flash memory will, however, differ depending on the available programming mode.

For devices using Byte Programming mode bytes are written with the "Write Program Flash Memory" command. This command sends a memory address ($**aa bb**) to select a 16-bit word, and selects low or high byte with the **H** bit (0 is low, 1 is high byte). The byte to be stored is then sent to the target AVR microcontroller in byte 4.

For devices using Page Programming mode the Flash is programmed in two steps. First a temporary Page buffer is filled using the "Load Program Memory Page" command. Each byte in this buffer can be directly accessed. Once the entire Page buffer is filled, it can be written to the Flash Memory using the "Write Program Memory Page" command.

In some devices, there is no method to detect when the Flash write cycle has ended. For this reason, the programmer presented in this application note waits N ms before attempting to send another command to the interface (the delay N will depend on target device, and can be found in the programming section of the datasheet). For some devices it is possible to use polling. When a byte is being programmed into the Flash or EEPROM, reading the addressed location being programmed will give a value M (often $FF). At the time the device is ready for a new byte, the programmed value will read correctly. This can be used to determine when the next byte can be written. When programming the value M polling will not work, and a delay N should be used before writing the next value. Polled mode will decrease the time required to program a device.

Table 7. Example, Reading "add r16,r17" as $0F01 From Flash Memory Location $104

Action	MOSI, Sent to Target AVR	MISO, Returned from Target AVR
Read $01 at address $104, low byte	$20 01 04 xx	$zz 20 01 **01**
Read $0F at address $104, high byte	$28 01 04 yy	$xx 28 01 **0F**

Table 8. Example, Writing "add r17,r18" as $0F12 to Flash Memory Location $10C (Byte Programming Mode)

Action	MOSI, Sent to Target AVR	MISO, Returned from Target AVR
Write $12 at address $10C, low byte	$60 01 0C **12**	$zz 60 01 0C
Wait N ms		
Write $0F at address $10C, high byte	$68 01 0C **0F**	$xx 68 01 0C
Wait N ms		

EEPROM Data Memory Access

Using the "Read EEPROM Data Memory" command, EEPROM contents can be read one byte at a time. The command sends a memory address ($**aa bb**) to select a byte location in the EEPROM.

Table 9. Example, Reading $ab From EEPROM Location $3F

Action	MOSI, Sent to Target AVR	MISO, Returned from Target AVR
Read $ab at address $3F	$A0 00 3F xx	$zz A0 00 **AB**

EEPROM is written one byte at a time, with the "Write EEPROM Memory" command. This command selects the byte to write just like "Read EEPROM Memory", and transfers the data to be written in the last byte sent to the target. For some devices there is no method to detect when the write cycle has ended. The programmer should simply wait N ms before attempting to send another command to the interface (the delay N will depend on target device, and can be found in the programming section of the datasheet). For increased programming speed, polling can be used as described in the "Flash Program Memory Access" section. An example of an EEPROM Write is shown in Table 10.

Table 10. Example, Writing $0F to EEPROM Location $11

Action	MOSI, Sent to Target AVR	MISO, Returned from Target AVR
Write $0F at address $11	$C0 00 11 **0F**	$zz C0 00 11
Wait N ms		

Lock Bits Access

To protect memory contents from being accidentally overwritten, or from unauthorized reading, the Lock bits can be set to protect the memory contents. As shown on Table 11, the memories can be either protected from further writing, or you may completely disable both reading and writing of memories on the chip.

In some devices the Lock bits can not be read, and setting Lock bits can not be verified by the programmer. To check that the Lock bits have been set properly in these devices, one should attempt to alter a location in EEPROM. When Lock bit 1 is set, memory locations are not altered. When both Lock bits 1 and 2 are set, no location can be read, and the result returned will be the Low byte of the address passed in the command. Setting only Lock bit 2 will have no protective effect. Before the chip is protected from reading, it has to be successfully protected from writing.

The Lock bits will only prevent the programming interface from altering memory contents. The core can read the Flash program memory and access the EEPROM as usual, independent of the Lock bit setting.

Table 11. Lock Bits Protection Modes

Lock Bit 1	Lock Bit 2	Protection Type
1	1	No Memory Lock
0	1	Further Programming of both Flash and EEPROM Disabled
0	0	Further Programming and Verification of both Flash and EEPROM Disabled

The only method to regain access to the memory after setting the lock bits, is by erasing the entire chip with a "Chip Erase" command. The lock bits will be cleared to 1, disabling the protection, only following a successful clearing of all memory locations.

On Chip Erase, the Lock bits obtain the value 1, indicating the bit is cleared. Although the operation of enabling the protection is referred to as "setting" the Lock bit, a zero value should be written to the bit to enable protection.

Table 12. Example, Setting Lock Bit 1 to Disable Further Programming

Action	MOSI, Sent to Target AVR	MISO, Returned from Target AVR
Set Lock Bit 1, Disable Programming	$AC FD xx yy	$zz AC FD xx
Wait N ms		

Chip Erase Operation   Before new contents can be written to the Flash Program Memory, the memory has to be erased. Without erasing, it is only possible to program bits in Flash memory to zero, not selectively setting a bit to one. Erasing the memory is performed with the "Chip Erase" command. This command will erase all memory contents, both Flash Program Memory and EEPROM.

Only after a successful erase of the memory, the Lock bits will be erased. This method ensures that data in the memories are kept secured until all data have been completely erased.

After a Chip Erase, all memory contents will be read as $FF.

The only way to end a Chip Erase cycle is by temporarily releasing the Reset line.

Table 13. Example, Erasing all Flash Program Memory and EEPROM Contents

Action	MOSI, Sent to Target AVR	MISO, Returned from Target AVR
Erase Chip	$AC 8x yy nn	$zz AC 8x yy
Wait N ms		
Release RESET to end the erase		

A Simple Low-cost In-System Programmer

This application note will not discuss all aspects of an In-System Programmer. Instead, it will show how a simple low-cost programmer can be made, using only an AT90S1200 and a few discrete components.

The programmer will plug into any serial port of any PC. The AT90S1200 doesn't come with a hardware UART, but the software will run a half duplex UART by using the Timer/Counter0 to clock data. The AT90S1200 also takes care of programming the target AVR by running the Master SPI entirely in software.

The schematics to the programmer can be seen in Figure 4. Power to the AT90S1200 is taken from the target system. The negative voltage needed to communicate serially with the PC is stored in C100 when receiving a logical one (negative line voltage).

The transmit line is fed with this negative voltage from C100, when transistor Q100 is closed. This sends a logical one on the transmit line. Logical zeros (positive voltage) is sent by opening Q100, connecting V_{CC} (actually V_{CC} - 0.2V) to the transmit line.

Some older PC systems might have serial port not accepting voltages below +10 volts as logical zero. This, however, is not a problem with the majority of existing PCs.

The file avr910.asm contains the firmware for the AT90S1200.

Figure 4. A Low-cost In-System Programmer
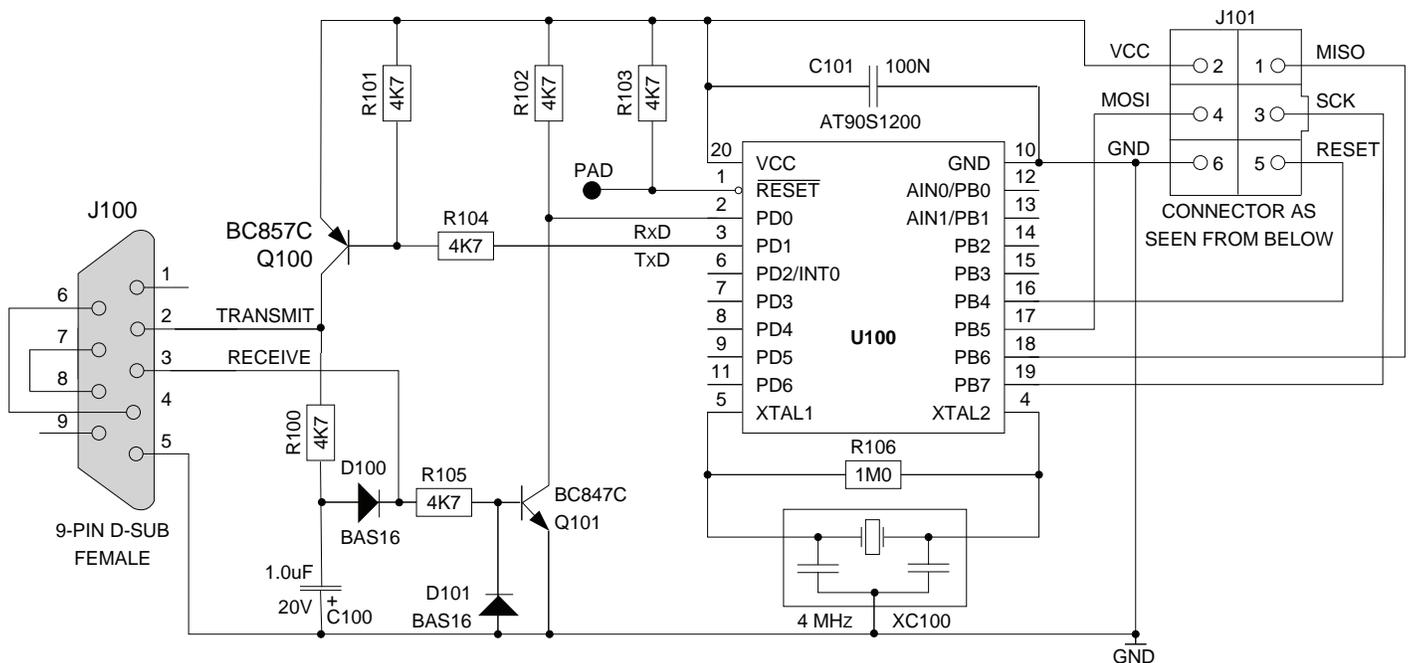
Part List

Table 14. Part List

QTY	Position	Value	Device	Tolerance	Vendor	Comment
1	C100	1U0/20V	CE1U020V	20%	PHILIPS +++	TANTAL CAPACITOR, SMD, (EIA3216)
1	C101	100N/50V	C08B100N	10%_X7R	MURATA +++	CERAMIC CAPACITOR, 0805, X7R
2	D100,D101	75V/100MA	BAS16		PHILIPS +++	SWITCH DIODE, SO-23 PACKAGE
1	J100	9 PIN	DSUB-9FSOL		HARTING +++	9 PIN D-SUB, FEMALE, SOLDER, 1.6 MM ROW SPACING, 2.54 MM PIN
1	JCABLE	6 PIN	HEADER6FC		HARTING +++	6 PIN HEADER (IDC), FEMALE, CABLE MOUNT
1	Q100	45V/100MA	BC857C		PHILIPS +++	SMD NPN TRANSISTOR, SO-23 PACKAGE
1	Q101	45V/100MA	BC847C		PHILIPS +++	SMD PNP TRANSISTOR, SO-23 PACKAGE
6	R100-105	4K7	R08_4K7	1%	KOA +++	RESISTOR, 0.125W, 1%, 0805
1	R106	1M0	NOT_USED	1%	KOA +++	RESISTOR, 0.125W, 1%, 0805
1	U100	SOIC-20	AT90S1200-4SC		ATMEL	AVR MICROCONTROLLER, 20 PIN SOIC
1	XC100	4.0MHZ	CSTCC4.00MG	0.5%	MURATA/AVX +++	CERAMIC RESONATOR, 4.00 MHZ, SMD (AVX: PRBC-4.0 B R)
1	HOUSING	9 PIN	D-SUB HOUSE	0.5%	AMP +++	9 PIN D-SUB PLASTIC HOUSING
1	CABLE	6 LEAD	FLATCABLE		HARTING +++	FLATCABLE, 6 LEAD, 300 MM
1	PCB	FR4/1.6MM	A9702.3.1000.A		ATMEL	PRINTED CIRCUIT BOARD NO. A9702.3.1000.A