AVR910: In-System Programming

Features

- **Program any AVR MCU In-System**
- **Reprogram both data Flash and parameter EEPROM memories**
- **Eliminate sockets**
- **Simple 3-wire SPI programming interface**

Introduction

In-System programming allows programming of any AVR MCU positioned inside the end system. Using a simple 3-wire SPI interface, the In-System programmer communicates serially with the AVR MCU, reprogramming all non-volatile memory 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 MCU to be programmed from any PC equipped with a regular 9-pin serial port.

The Programming Interface

For In-System programming, the programmer is connected to the target using as few wires as possible. To program any AVR MCU in any target system, a simple 6-wire interface is used to connect the programmer to the target PCB. Figure 1 below shows the connections needed.

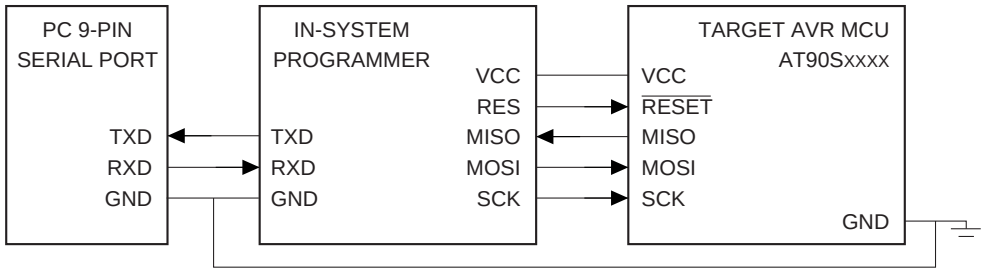

Figure 1. 6-wire connection between programmer and target system.

The Serial Peripheral Interface (SPI) interface consist of the 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.

8-Bit AVR Microcontroller

Application Note

0943B-B–10/97

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 MCU reset 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 MCU reset line to automate this process using a fourth control line. (RES)

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.

Table 1. Connections required for In-System programming

Pin	Name	Comment
SCK	Shift 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.
RES	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 power through the In-System programming connector for the duration of the programming cycle.

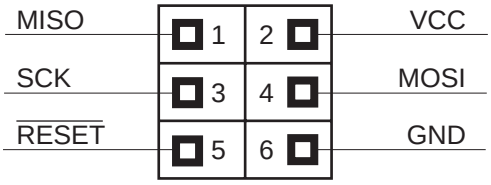

Figure 2. The Recommended In-System Programming Interface Connector Layout.

Figure 2 shows the connector used by all Atmel In-System programmers to connect to the target system. The standard connector supplied is a 2x3 pin header contact, with pin spacing of 100 mils.

Hardware Design Considerations

To allow In-System programming of the AVR MCU, the pins of the target MCU need to be released by the target system on request. This chapter 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 MCU 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, active reset must immediately disable any chip driving these lines from the target system. Note that the AVR MCU will automatically set all its I/O pins to inputs, with pull-ups disabled, when RESET is active.

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.

If using the In-System programmer to control the reset of the target system is impossible, the reset can be controlled manually. The programmer would have to request the operator to apply reset when this is required. The operation will be handled safer and faster if the In-System programmer is allowed to read the target AVR reset line to verify the user performs the requested tasks. With a change in software, the programmers supplied from Atmel can support this lack of reset control. However, this method is not recommended.

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 synchronization is lost, the only means of regaining synchronization is to release the RESET line for more than 100 ms.

The target AVR MCU 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 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 MCU 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 MCU, 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 MCU 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 MCU.

V_{CC}

When programming the target MCU, 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 20mA. The programmer supplied from Atmel operate 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 MCU is reset is active, this line should never be driven by the target system. **Edges on this line after RESET is pulled low will be critical, and cause the target AVR MCU 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 MCU is reset is active, this line should never be driven by the target system. When programming, oscillations on this pin should be tolerated by the surrounding system when the AVR reset is active.
MISO	When the target AVR MCU is 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 wary depending on the programmer being used.

Programming Protocol

Immediately after RESET goes active on the target AVR MCU, 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. **It is very important to keep the SCK pin stable, as one single edge will cause the target to loose synchronization with the programmer.** After pulling reset low, wait at least 20 ms before issuing the first command.

COMMAND FORMAT

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

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

Action	MOSI, Sent to target AVR	MISO, Returned from target AVR
Programming Enable	$9C 53 xx yy	$zz 9C 53 xx
Read Device Code $1E at address $00	$30 nn 00 mm	$yy 30 nn **1E**

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 always sent on MOSI and MISO lines with most significant bit (MSB) first.

Table 4. Serial Programming Command Format

Instruction	Instruction Format				Operation
	Byte 1	Byte 2	Byte 3	Byte4	
Programming Enable	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	Enable Serial Programming after $\overline{\text{RESET}}$ goes low
Chip Erase	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	Chip erase both Flash, EEPROM and lock-bits
Read Program Flash Memory	0010 **H**000	0000 **aaaa**	**bbbb bbbb**	**oooo oooo**	Read **H**(high or low) data **o** from Program memory at word address **a:b**
Write Program Flash Memory	0110 **H**000	0000 **aaaa**	**bbbb bbbb**	**iiii iiii**	Write **H**(high or low) data **i** to Program memory at word address **a:b**
Read EEPROM Memory	1010 0000	0000 0000	**bbbb bbbb**	**oooo oooo**	Read data **o** from EEPROM memory at address **b**
Write EEPROM Memory	1110 0000	0000 0000	**bbbb bbbb**	**iiii iiii**	Write data **i** to EEPROM memory at address **b**
Write Lock Bits	1010 1100	111x x***21***x	xxxx xxxx	xxxx xxxx	Write lock bits. Set bits ***1,2***='0' to program lock bits
Read Device Code	0011 0000	xxxx xxxx	0000 00**bb**	**oooo oooo**	Read Device Code **o**

Note: **a** = address high bits
b = address low bits
H = 0 - Low byte, 1 - High Byte
o = data out
i = data in
x = don't care
1 = lock bit 1
2 = lock bit 2

ENABLE MEMORY ACCESS

When the RESET pin is first pulled active, the only instruction accepted by the SPI interface is a "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 above shows an example where memory access is enabled in the first command sent to the chip.

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 MCU will not respond with an acknowledge to the “Programming Enable” command. To check the command has been accepted by the target AVR MCU, we could try to read the Device Code, 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, also known as the signature bytes. The device code will identify the chip vendor (Atmel), the part family (AVR), flash size in kB, and family member (ex. AT90S1200). The “Read Device Code” command format is found in Table 4: Serial Programming Command Format Table 4 to be [$30, $XX, $adr, $code]. Valid addresses are $0, $1 or $2. Table 5 shows what the expected result will be

.

Table 5. 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	$90 indicates AVR with 1kB Flash memory $91 indicates AVR with 2kB Flash memory $92 indicates AVR with 4kB Flash memory $93 indicates AVR with 8kB Flash memory
$02	Part Number	Identifies the part, see Table 6 for details

Table 6. Part Number Identification Table

Part Family and Flash Size	Part Number	Part
$90	$01	AT90S1200
$92	$01	AT90S2313
$93	$01	AT90S4414
$94	$01	AT90S8515
$FF	$FF	Device Code Erased (or Target Missing)
$01	$02	Device Locked

Table 6 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 and the MISO line is constantly pulled high. The programmer can detect this situation by detecting that also commands 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 7. 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 Fam. and mem 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. Using the “Read Flash Program Memory” command, Flash memory contents can be read one byte at a time. The command sends a memory address ($0a 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 stored at this address is then returned from the target AVR MCU 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 8.

Table 8. 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**

Flash memory is written with the “Write Program Flash Memory”. The command sends a memory address ($0**a 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 MCU in byte 4.

Unlike parallel programming, there is no method to detect when the Flash write cycle has ended. The programmer should simply wait 4 ms before attempting to send another command to the interface. If a command is sent before the write cycle has ended internally, the write might be disturbed and invalidated.

Table 9. Example, Writing 'add r17,r18' as $0F12 to Flash Memory location $10C

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 4 ms		
Write $0F at address $10C, high byte	$68 01 0C **0F**	$xx 68 01 0C
Wait 4 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 ($0**a bb**) to select a byte location in the EEPROM.

Table 10. 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 like Flash memory, with the “Write EEPROM Memory” command. This command selects byte to write just like “Read EEPROM Memory”, and transfers the data to be written in the last byte sent to the target. Unlike parallel programming, there is no method to detect when the Flash write cycle has ended. The programmer should simply wait 4 ms before attempting to send another command to the interface. If a command is sent before the write cycle has ended internally, the write might be disturbed and invalidated.

Table 11. 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 4 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 seen from Table 12 below, the memories be either protected from writing by programming, or the programming interface can be disconnected completely from the memory block, disabling both reading and writing of memories on the chip.

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, 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 both 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 12. 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 protecting.

Table 13. 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 4 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.

After successfully erasing the memory, the lock bits are erased. This method ensures that data the memories are kept secured until all data have been completely erased before access is again granted.

Following a chip erase, all memory contents will read as $FF.

The only way to end a chip erase cycle, is by temporarily releasing the RESET line.

Table 14. 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 10 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. This programmer can be obtained from Atmel, or one of Atmel's distributors.

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/Counter 0 to clock data. The S1200 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 3 below. Power to the S1200 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} - 0.2 V) 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.

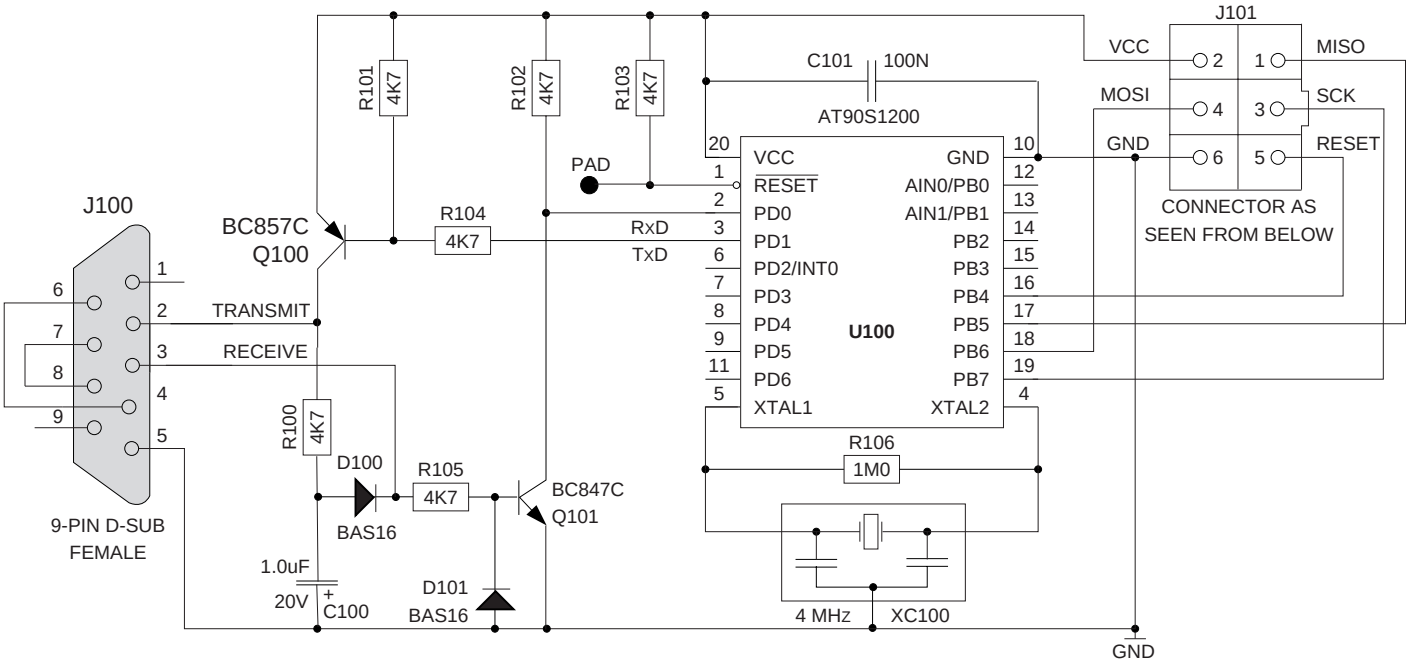

Figure 3. A Low Cost In-System Programmer

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, 1.6MM 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