TinyLoader/B r1.0

A minimal-cost loader for AVR microcontrollers

© 2006 Kasper Pedersen
tinyloader@kasperkp.dk
http://tinyloader.kasperkp.dk

Overview

Microcontrollers such as ATTiny13/24/45/85 look good on paper; Only eight pins, and tiny little SMT packages. Perfect for that clock divider or egg timer functionality. The only problem is: To reprogram them one needs to provide a programming connector, it needs 7 wires for HV programming mode, and the HV programmer is not a five minute job.

TinyLoader is a boot loader for ATTiny that attempts to solve this problem. It provides a simple serial programming interface through a single pin, and can be interfaced to a PC with two passive components.

Code size is 48 words.

Works from the on-chip oscillator.

Allows one to use the reset pin as I/O without having a HV programmer at hand.

A Windows program is provided for talking to the loader.

Bird's eye view of the functionality

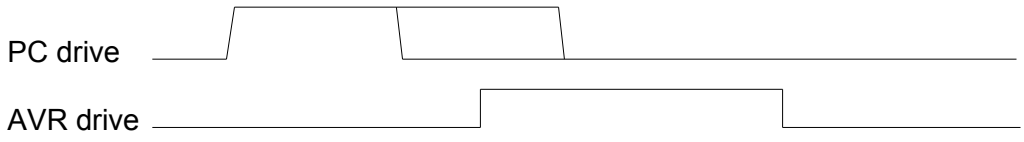
Since the loader has to fit into an extremely small footprint, all intelligence is at the host end. The loader just receives 24 bit packets, executes them, and returns the result to the host. To further simplify implementation, the link is synchronous duplex: When the host sends a packet, it also gets the reply to the previous command back.

On power up the loader has to decide either to stay in loader, or execute the main program. The loader listens, and if the first thing it receives is a specific 24-bit packet, it stays in the loader and receives a new flash image from the PC. If the packet doesn't match, or the receiver times out, it starts the main program.

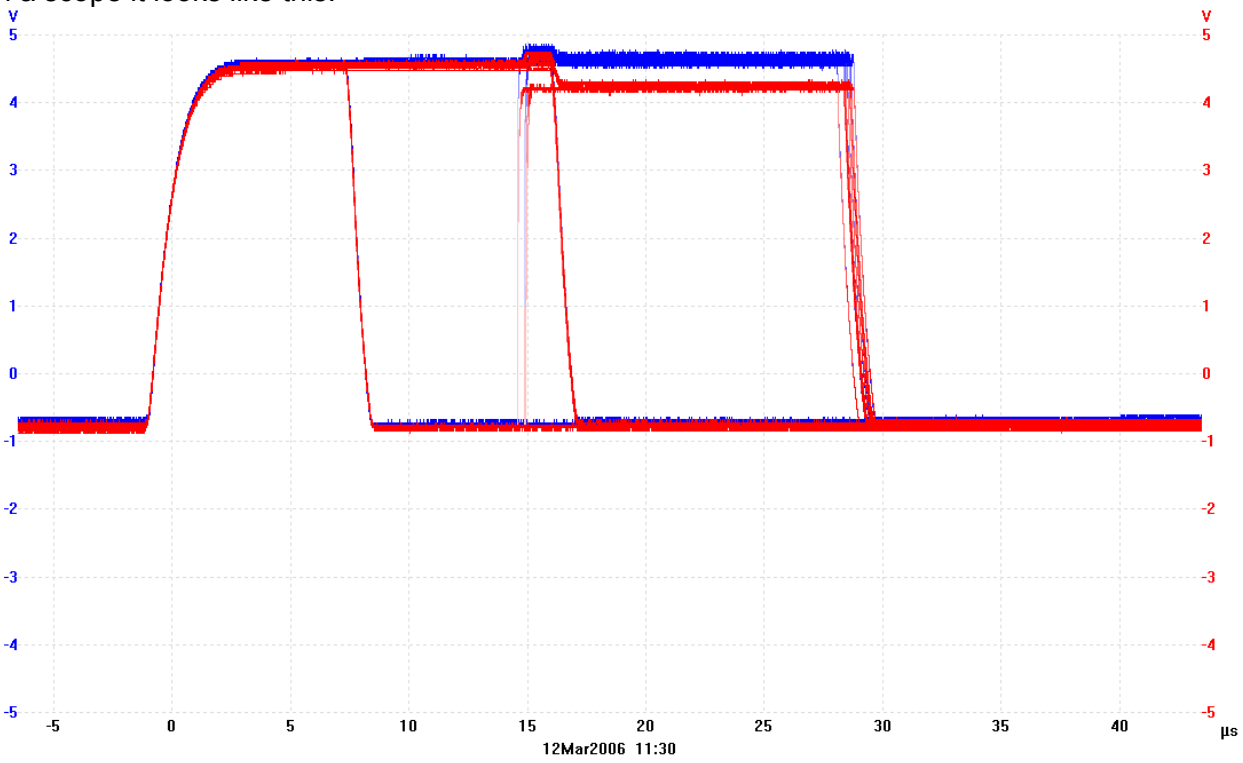
Physical layer protocol

A fancy way to say “what happens on the wire.”

The PC initiates every bit transfer by sending a pulse. If the pulse is 1T long its a 1, and if it's 2T long it's a 0. The loader samples the bit at T=1.6, and then drives a pule of its own, lasting another 1.6T. Since this is really the same wire, it means that the PC provides the start bit and the first bit, and the loader provides the second bit in the serial frame. The result is that this is directly compatible with a standard serial port, but doesn't require the AVR to have precise timing.



On a scope it looks like this:



Here the red trace is at the PC end, and the blue trace is at the AVR end, with a 100 ohm resistor in between so we can see when the AVR is overriding the bus.
From 0 to 8 microseconds is the PC's startbit, followed by either a high or a low bit. The AVR replies with its own bit after sampling the PC's. This is seen as the red and blue trace being different since the AVR is sourcing current.

A total of 24 bits is transferred, most significant bit first. Bits 24..8 are data or address, and bits 7..0 are instruction.

PC interface

The interface used here is a 4k7 resistor from TXD to RXD, a 5.6V zener from RXD to GND, and a two meter long twisted pair with GND and RXD connecting it to the Tiny13. On the AVR end there's a 100 ohm resistor in series with the cable. Do not omit this 100 ohm resistor or you'll get nasty reflections. Oh, and there's a disclaimer about this at the end of the document.

Instruction set

The instruction set depends on the AVR used since there is almost no translation between the wire protocol and the SPMCR register.
Replies are sent back on the next instruction.

For Tiny13/25/45/85 the useful ones are:

aa bb 00 FLASHREAD
aabb is the flash address for the read operation. The reply takes the form cc dd F0.
Example:

sent: 12 34 00	received: xx xx F0	we send an address of 1234
sent: 12 36 00	received: AB CD F0	the result is ABCD, the F0 is a sync byte.
sent: 00 00 00	received: EF 01 F0	and yet another word read.

The addresses are byte addresses. The first byte is fetched from address +1, the second byte is fetched from address +0.

aa bb 08 EEREAD
As FLASHREAD, only reads a single byte from EEPROM
Example:

sent: 00 34 08	received: xx xx F0	we send an address of 34
sent: 00 00 00	received: xx CD F0	the result is CD, the F0 is a sync byte.

aa bb 00 SET_POINTER
A side effect of FLASHREAD is to set the page pointer used for write operations. The value loaded into the pointer is two higher than the address given, so if you want to set the pointer to the address of 0000, the parameter aabb should be FFFE.

00 00 11 CLEAR_PAGE_BUFFER
Fills the flash buffer with ones in preparation for a page load. Side effect: Increments the write pointer by 2.

aa bb 01 FILL_PAGE_BUFFER
transfers the contents to the page buffer, indexed by the write pointer. Increments the write pointer by 2.
Example:

sent: FF FE 00	received: xx xx F0	sets the write pointer to 0000
sent: 34 12 01	received: xx xx F0	write 12 34 to address 0000 and 0001
sent: 78 56 01	received: xx xx F0	write 56 78 to address 0002 and 0003

00 00 03 PAGE_ERASE
Erases the page pointed to by the write pointer.
Example:

sent: FF FE 00	received: xx xx F0	sets the write pointer to 0000
sent: 00 00 03	received: xx xx F0	erase the first page.

Then wait the maximum erase time.

00 00 05 PAGE_WRITE
Programs the page pointed to by the write pointer. Increments the write pointer by 2.
Example:

sent: FF FE 00	received: xx xx F0	sets the write pointer to 0000
sent: 00 00 05	received: xx xx F0	program the first page.

Then wait the maximum write time.

Notice that, because there wasn't room, the loader does not include EEPROM write capability.

Procedure for entering the loader

With the AVR powered off, the PC sends out one 24-bit 'magic' frame every 100 milliseconds:

17 00 6A

The AVR is then powered on, and for approximately 200 milliseconds it will listen for this frame. If the first frame received is this exact sequence, it enters the bootloader. The frame sent to the PC as the very first frame is

00 00 F0

When the PC receives this reply, it stops sending the magic frame, and proceeds to verify the integrity of the link. This is done by reading the last word of flash

Sent: **FF FE 00** received: **xx xx F0**
Sent: **FF FE 00** received: **08 95 F0**
Sent: **FF FE 00** received: **08 95 F0**
Sent: **FF FE 00** received: **08 95 F0**

as many times as can fit in 2 seconds. This is done to give the user time to *properly* switch the power supply.

The '17' and '6A' are user configurable within limits: The first number is also the loop count for receive timeout, and the last must be even to prevent it being interpreted as a write.
The value 0x17=23 dec. corresponds to a one second timeout value @9.6MHz clock.

In version 1.0 the second byte is also user definable in the source code.

Pitfalls

If you write a host application, send packets continuously: If you delay for more than the session timeout (the 0x17), it will exit the loader and branch to the main program.
If you need to delay, send flash reads as NOPs.

2 times out of 100, when one tries to activate the loader, it will fail. This happens if the loader powers up in the 2 millisecond window where the PC is transmitting the magic frame, since the PC and loader will be out of synchronization. This was a design choice.

Customizing the loader

In the source to the loader, there are a few interesting parameters.

asytimeout=4

Specifies how long to listen for the magic frame. It is given in units of 7*65536 clocks, so a value of 4 gives 4*7*65536/9.6MHz = 0.191 seconds.
Tune this to suit the clock rate.

asymagic1=0x17 ;23

Specifies how long to wait for normal commands once the magic packet has been received. It is given in units of 7*65536 clocks, so a value of 23 gives 23*7*65536/9.6MHz = 1.1 seconds.
Tune this to suit the clock rate, 1-2 seconds is good. If you have multiple nodes on the same wire, it can be used as a node address.

asymagic3=0x6A

This can be used as a node address; It must be even, otherwise it will be interpreted as a write, and that wouldn't be good.

asyptimer=39

This constant sets the baud rate. The delay of (14+3*asyptimer)/Fcpu should give approximately 1.6 bit times.
Example:

At 115200 baud (the default loader rate), 1.6 bit is 1.6/115200=13.9 microseconds.
We set the clock frequency to 9.6MHz. Solving for asytimer gives us:

asytimer = (13.9µs*9.6MHz-14)/3 = 39.77 => 39.

and for 8MHz 31 is a good value.

If you use the value for 9.6MHz with 4.8MHz clock, you'll just have to tell the PC to run at half baudrate.

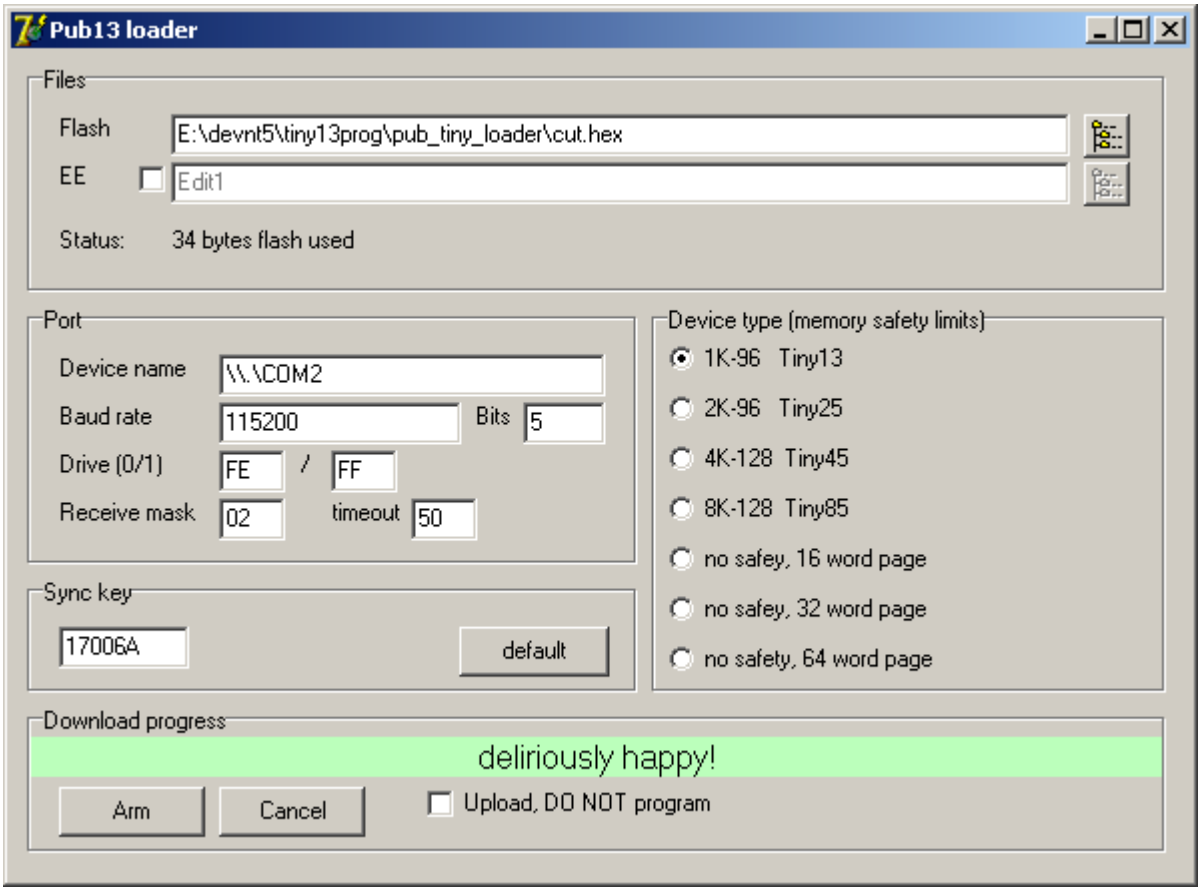
asybitno = 4
asypin = PINB
asyport = PINB
asyddr = DDRB

Defines the port pin on which the loader operates.

asyidlepol = 0

Defines wether it's a TinyLoader/B (=0) with RS232 polarity, or a TinyLoader/A (=1) with TTL polarity.
You'll most likely use /B with the resistor-zener interface, and /A with a MAX232 interface (described at the end of this document)

The PC program



It's written in Delphi 7, and under the hood it's quick-and-dirty-simple.

- Select a file for Flash. It accepts Intel hex, Motorola Srecord, or binary files.
- Check that Status reports a sane value.
- Modify the sync key if you have modified the loader
- Arm it.

Once armed it will continuously transmit magic frames. Power on the AVR, and it should download to it. There are a few things that can go wrong:

Bad response from loader
: The sync key provided isn't correct.

First instruction looks like it's incorrect
: If the very first instruction in flash isn't a jump to bootloader, it will lock you out. Thus, change the first instruction to be a rjmp to bootloader, and use the ASYmainjump softvector instead.

The device would not accept the parameters
: This happens when a USB serial port won't run 5-bit frames. Try 7 or 8 bit frames instead.

Flash is too big to fit!
: Self explanatory. Check that you have selected the correct device type. The check is there to keep you from overwriting the boot loader.

It just doesn't work!
: Since the PC is driving into the AVR's protection diodes, it may be able to provide enough current for the AVR to start if you've set the BOD to a low value (measure VCC to see if it is the case). In that case, put 470 ohms across VCC-GND and try again. Or if you're brutal, just power up your device, then briefly short the power supply with a suitably small resistor (you need a few ohms so as not to cause nasty inductive spikes). If you haven't disabled the reset pin, you can use that as an alternative.

The code

```
.include "tn13def.inc"

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;  Project name         TinyLoader/B
;
;  Descripion           48-instruction one-wire boot
;                       loader for ATTiny13/24/45/85
;
;  Author               Kasper Pedersen
;
;  Contact              tinyloader@kasperkp.dk
;
;  Copyright            2006 Kasper Pedersen, all rights reserved.
;                       Permission given to use, modify, and re-
;                       distribute, provided that redistributions
;                       include source and does not misstate the
;                       origins of this code.
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;
;  Here's an install guide!
;
;  Tiny13 pinout: if we run the loader on pin 3
;
;                 _______________
;        vpp__|                 |__
;          |__| PB5      VCC  |__|
;           1 |                 | 8
;        sci__|                 |__sdo
;          |__| PB3      PB2  |__|
;           2 |                 | 7
;           __|                 |__sii
;   link<< |__| PB4      PB1  |__|
;           3 |                 | 6
;           __|                 |__sdi
;    gnd<< |__| GND      PB0  |__|
;           4 |_________________| 5
;
;
;  We must specify that. The configuration is here:
;
.equ asybitno   = 4       ;B4
.equ asypin     = PINB
.equ asyport    = PORTB
.equ asyddr     = DDRB
.equ asyidlepol = 0       ;0=AsyLoader/B  1=AsyLoader/A
;
;  And depending on the clock frequency you may have to change the
;  bit timing value here. There's a formula in the PDF.
;
.equ asyptimer  = 39      ;39=9.6MHz 17=4.8MHz 30=8MHz
```

```
;
; Then specify how patient the loader is in the pick-up phase. The
; value is given in units of 458752 clocks. 4 gives a suitable
; 200 ms window at 9.6MHz. For 4.8MHz you want to use 2 for quicker
; startup, and at 20MHz 8 is good. Rule of thumb: You need to have
; 170+ milliseconds!
;
.equ asytimeout = 4    ;47-ms slices to wait for sync frame.
;
; Then once it's in the loader, there's another timeout before it
; skips out to main. The default values are rather universal.
; The only time one might want to change them is if one
; runs at very low clock speeds, such as 110kHz, and want the
; auto-start-after-download functionality to work.
; Then modify asymagic1, it's given in the same units as
; asytimeout, and you want 1+ second.
; asymagic3 can be used as a node address.
;
.equ asymagic1  = 0x17 ;also used as post-sync timeout value
.equ asymagic2  = 0x00 ;hardcoded, ONLY here for reference
.equ asymagic3  = 0x6A ;must not translate to a write instruction
;
; Then there's the program. The very first instruction must be an
; rjmp to bootloader.
;
.org 0
        rjmp TinyAsyLoad        ;for Tiny 13,this is the vector area.
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
.org 0x0A
        nop
        nop         ;and this is unused, residing in the first page.
        nop
        nop
        nop
        nop
;
; Since this is a demo and my code is small, my program does not
; use the first flash page. When the first flash page is unchanged,
; the PC loader won't do an erase/write cycle on the first page.
;
; And here's main.
;

.org 0x10
main:
     ;   sbi PORTB,0  ;Using a T13 as a 1/8 clock divider :-)
     ;   sbi DDRB,0
     ;   cbi PORTB,0
        rjmp main
```

```
;
; The user reset vector is located at the end of user flash.
;
.org 0x1CF
ASYmainjump:
        rjmp main
;
; and then comes the loader.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;Maximally Ugly Resident Loader for T13 (C) 2006 Kasper
Pedersen;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.equ eereadoption=1
;;;;;;;;;;;;;;;;;;;;;;;; settings for COMMS polarity loader.
.IF asyidlepol==0
 .MACRO ASYskip_if_driven
        sbis  asypin,asybitno
 .ENDMACRO
 .MACRO ASYskip_if_not_driven
        sbic  asypin,asybitno
 .ENDMACRO
 .MACRO ASYdrive_sig
        sbi   asyport,asybitno
        sbi   asyddr,asybitno
 .ENDMACRO
 .MACRO ASYrelease_sig
        cbi   asyddr,asybitno
        cbi   asyport,asybitno
 .ENDMACRO
.ENDIF
;;;;;;;;;;;;;;;;;;;;;;;; settings for 1WIRE polarity loader.
.IF asyidlepol==1
 .MACRO ASYskip_if_driven
        sbic  asypin,asybitno
 .ENDMACRO
 .MACRO ASYskip_if_not_driven
        sbis  asypin,asybitno
 .ENDMACRO
 .MACRO ASYdrive_sig
        cbi   asyport,asybitno
        sbi   asyddr,asybitno
 .ENDMACRO
 .MACRO ASYrelease_sig
        cbi   asyddr,asybitno
        cbi   asyport,asybitno
 .ENDMACRO
.ENDIF



.IF asymagic2==0
  .DEF as2reg=r17
.ELSE
  .DEF as2reg=r22
.ENDIF
```

```
; sssssssss000000000111111111222222222333333333444444444555555555566666666677777777eeeeeeeee
; (         ^          ^-----------)   (          ^          ^-----------)
; we can drive 2 bits per byteframe, so 22kbps datarate.
; with autoinc that means a transfer rate of 2k/second. yay! speedy!
;
; T=0.0  start bit
; T=1.0  host drives data bit for T13
; T=1.6  T13 samples, and drives own data
; T=2.0  host removes data
; T=2.5  host reads data
; T=3.2  T13 removes data
; T=4.8  T13 ready
;
;  r0, r1, r16  comms register
;  r17          bit counter
;  r24,r25      timeout divider
;  r20          timeout highpart
;  r21          delay counter
;
.org 0x1D0
;;;;;;;;;;;;;;;; 0x1D0 ;;;;;;;;;;;;;;;
TinyAsyLoad:
        clr r0 ;prevent drive for the first 16 cycles.
        clr r1
.IF asymagic2==0
        nop
.ELSE
        ldi as2reg, asymagic2
.ENDIF
        ldi  r20,asytimeout+1  ;200-250 ms timeout for programming
startup
ASYloader:
        ldi  r16,0xF0          ;frame indicator
        ldi  r17,24            ;bit count

ASYbit: sbiw r24,1             ;this operates as a /65536 prescaler
        sbci r20,0             ; <- timeout counter. 7/loop
        breq ASYmainjump       ;timeout -> start.
        ASYskip_if_driven      ;wait for startbit.
        rjmp ASYbit
        rcall delay_1p5bit
        add   r16,r16
        ASYskip_if_driven      ;sample inat +10+n*3+[6]
        inc   r16
        adc   r0,r0
        adc   r1,r1
        brcc  ASYover
        ASYdrive_sig           ;drive if the bit is clear. +16+n*3+[6]
ASYover:
        rcall delay_1p5bit
        ASYrelease_sig         ;drive if the bit is clear
        rcall delay_1p5bit     ;release-edge-fix
        dec   r17
        brne  ASYbit

        ;we have a full frame. The first frame is a magic unlock frame.
        ;the first command must be 17 00 6A
        ldi   r20,asymagic1    ;reset the timeout value to a large value
```

```
        brts  ASYexecute
        cpi   r16,asymagic3    ;magic address
        cpc   r0,as2reg        ;reusing the expired bit counter (opt)
        cpc   r1,r20           ;reusing the timeout as key byte
        brne  ASYmainjump      ;otherwise bail out
        set
ASYexecute:
        sbrs  r16,0            ;if it's a read, there is addressing
information.
        movw  r30,r0
        ;sbrs  r16,3            ;read fuse and lock bits option
        out   SPMCSR,r16       ;will have no effect when bit0 isn't set.
        spm
.IF eereadoption==1
        out   EEARL,r30        ;eeprom read option: set the fuse-read to
        sbi   EECR,EERE        ;read out the eeprom
        in    r0,EEDR
        sbrs  r16,3
.ENDIF
        lpm   r0,Z+            ;flash read when invalid instruction
        lpm   r1,Z+            ;autoincrement for free at the same time
        rjmp  ASYloader

delay_1p5bit:
        ldi   r21,asyptimer
delayloop1:
        subi  r21,1            ;1
        brne  delayloop1       ;2  3*r18_initial
        ret                    ;4+3 call   THIS MUST RESIDE AT THE LAST
;200                                        ADDRESS OF FLASH
```

Disclaimer regarding the interface circuit

If you're reading this part, you probably think the interface circuit is a little bit crappy.
You're not alone.

But, apparently, that is what the world wants. Shitty Taiwanese designs that break as fast as they can push them over the counter sell like hotcakes.

Design your own. A MAX232 and a couple of inverters come to mind. Or configure it to be a TinyLoader/A to save the inverter.

If you are Taiwanese and make industry-quality products, apologies.

TinyLoader/A

If you define asyidlepol=1 in the source, it will be compatible with a 1-wire interface. That means that if you have a MAX232 with one resistor between the TTL level TX and RX pins, you have a very nice, robust interface.

It also means that it's compatible with my 1-Wire interface (1Wire is a Dallas-Maxim trademark for their one-signal-wire rom/eeprom/..family)