The debugWire protocol

Connecting to the dW bus

debugWire use the same format as rs232, the default baudrate is clock/128.
Using an ATmega32U2 I obtained many dumps of the protocol to try to decipher it, turns out it was easier than expected.

Here is how I did it:

Connect the target to a 16MHz crystal and clear CKDIV8 then connect a Dragon/JtagIce.
Connect the RX line of an ATmega32U2 with dumping FW to the dW/reset line and set the baud to 125kbps.

UNTESTED
Connect the target to a 14.745MHz crystal and clear CKDIV8 then connect a Dragon/JtagIce.
Connect the RX line of a PC COM port via a level converter to the dW/reset line and set the baud to 115200bps.
UNTESTED

For interactive access:
Disconnect the Dragon/JtagIce first.
The TX line can then be connected via a transistor (open collector).
Connect the TX line to the emitter, VCC via a pullup to the base and VCC via another pullup to the collector.
10kOhm seems to work fine. Then use HTerm or something similar.

Known debugWire commands. All commands are given in hex.

It seems that the commands fall into four groups:
0x 1x 2x 3x is mostly for flow control.
4x 5x 6x 7x seems to be for setting flags, something like BCR on the JTAG OCD.
8x 9x Ax Bx is for baudrate control.
Cx Dx Ex Fx provides access to the PC, BreakPoint, Intruction, and Signature registers.

To sync issue a break, this will also stop the target.
When there is a collision, the target issues a break.
Seems that the target issues a break when hitting a breakpoint.
All breaks are followed by 55 from the target, and the baud is reset to clock/128.
The 55 is used by the debugger (Dragon/JtagIce) to detect the baud rate.

Flow control

06 -- Disable dW, this enables ISP.
07 -- Reset, this is followed by a break and 55.

20 -- GO when reading/writing memory
21 -- SS when ?reading?/writing a single register
22 -- not seen yet --
23 -- SS when executing an instruction loaded with D2
30 -- GO resume normal exection
31 -- SS seems to be SingleStep, the PC will increment twice.

32 -- not seen yet --
33 -- not seen yet --

Setting the flags

It seems that all commands starting with 01xx xxxx sets the flags.

60 -- used before a GO
61 -- used before a Run to Cursor
63 -- used before a Step Out
79 -- used before a Step In and Auto Step
79 -- used before resuming a SW BP
7A -- used before a SingleStep
66 -- used before Reading/Writing memory
64 -- seen in Write Flash Page
44 -- seen in Write Flash Page

It seems that bit 5 = 0 when "Run Timers" is set.
40 -- used instead of 60 when "Run Timers" is set
41 -- used instead of 61 when "Run Timers" is set
43 -- used instead of 63 when "Run Timers" is set --not seen yet--
46 -- used instead of 66 when "Run Timers" is set
59 -- used instead of 79 when "Run Timers" is set
5A -- used instead of 7A when "Run Timers" is set

Setting the baudrate

The following commands is used to set the baud rate to 125kbps. After setting the baud 0x55 is returned.
0x83 clk/128 16MHz -- Don't know why this is used, baud rate stays the same.
0x82 clk/64 8MHz
0x81 clk/32 ? 4MHz --not seen yet--
0x80 clk/16 2MHz
0xA0 clk/8 1MHz
0xA1 clk/4 ? --not seen yet--
0xA2 clk/2 ? --not seen yet--
0xA3 clk/1 ? --not seen yet--

The registers

Cx ll -- will set the low byte.
Ex ll -- will get the low byte.

D0 hh ll -- set the PC to hhll
D1 hh ll -- set the HW BreakPoint to hhll
D2 hh ll -- set the instruction register to hhll
D3 hh ll -- Steffanx got lucky, this didn't change his signatures. :)

The following reads the registers and returns hhll from the target.
F0 hh ll -- get the PC. TAKE NOTE after a break this will return PC+1.
F1 hh ll -- returns the previously set BP.
F2 hh ll -- returns the previously set instruction.
F3 hh ll -- returns the signature (dW ID).

According to davidc__ the order of D0 D1 and C2 is not important. Steffanx dicovered that the commands repeat eg: F0 = F4 = F8 = FC.
This implies that only bits 0+1 are decoded and bits 2+3 are ignored.
This is true for all the register commands.
Reset also repeats 07 = 0F = 17 = 1F = 27 = 2F = 37 = 3F.
So does disable dW 06 = 0E = 16 = 1E = 26 = 2E = 36 = 3E.

Practical use of the commands as seen in the dumps.

Start of debugging

00 -> 55 -- the 00 is a break
83 -> 55 -- set the baud
f3 -> 94 89 -- 16u2 signature
00 -> 55
07 -> 00 55
83 -> 55
F0 -> 00 01 --PC = 0000--

End of debugging

00 -> 55
07 -> 00 55
83 -> 55
F0 -> 00 01 --PC = 0000--
D0 00 00 60
D0 00 00 30 --GO--

Resuming execution

D0 00 00 xx -- set PC, xx = 40/60 - 41/61 - 59/79 - 5A/7A
D1 00 01 -- set breakpoint (single step in this case)
D0 00 00 30 -- set PC and GO

Resuming from a SW BP
D0 00 00 79/59 -- set PC
D1 00 01 -- set breakpoint (single step in this case)
D2 ii ii -- load the instruction replaced by the break.
D0 00 00 32 -- set PC and GO

Step Out -- D1 isn't used
D0 00 00 63/43 -- set PC
D0 00 00 30 -- set PC and GO

Executing an instruction

D2 hh ll 23 -- hhll is the hex code for the instruction.
Seems that its not possible to execute a 32 bit instruction this way.
The Dragon reflash the page to remove the SW BP, SS and then reflash again with the SW BP!!!

Selecting the Read/Write mode

C2 xx
00 = Read SRAM
01 = Read Registers
02 = Read Flash
04 = Write SRAM
05 = Write Registers

Reading the registers

66 D0 00 00 D1 00 20 C2 01 20 -> target returns 32 bytes

66
D0 00 00 -- Set the register to start at.
D1 00 20 -- Set the register to stop at + 1.
C2 01 -- Set mode to: read registers.
20 -- GO, start reading.
--32 bytes from target--

0000: out DWDR,r0 ----> 001F: out DWDR,r31 = 1F

Writing Registers

66 D0 00 00 D1 00 20 C2 05 20 --32 bytes--

66
D0 00 00 -- Set the register to start at.
D1 00 20 -- Set the register to stop at + 1.
C2 05 -- Set mode to: write registers.
20 -- GO, start writing.
--32 bytes to target--

Write a single register
66 D0 00 r D1 00 r+1 C2 05 21 xx -- TAKE NOTE: use 21 not 20.
Not sure if D1 is actually used since 21 seems to be SS.

0000: in r0,DWDR ----> 001F: in r31,DWDR

Reading SRAM

66 D0 00 1E D1 00 20 C2 05 20 ll hh D0 00 00 C2 00 D1 00 02 20 xx

66
D0 00 1E D1 00 20 C2 05 20 ll hh -- Z = hhll

D0 00 00
C2 00 -- Read SRAM
D1 00 02
20 -- GO, start reading.
xx -- byte from target

C0 00 20 xx will read the next location.

0000: ld r16,Z+
0001: out DWDR,r16

Writing SRAM

66 D0 00 1E D1 00 20 C2 05 20 ll hh C2 04 D0 00 01 D1 00 03 20 xx

66 D0 00 1E D1 00 20 C2 05 20 ll hh -- Z = hhll

C2 04 -- Write SRAM
D0 00 01
D1 00 03
20 -- GO, start writing.
xx -- byte to target.

C0 01 20 xx will write the next location ???

0001: in r16,DWDR
0002: st Z+,r16

Reading a Flash Page

66 D0 00 1E D1 00 20 C2 05 20 ll hh D0 00 00 C2 02 D1 01 00 20 --128 bytes--

66
D0 00 1E D1 00 20 C2 05 20 ll hh -- Z = hhll

D0 00 00
C2 02 -- Read Flash
D1 01 00
20 -- GO, start reading.
--128 bytes from target--

0000: lpm r?,Z+
0001: out DWDR,r?

Writing a Flash Page

66 D0 00 1A D1 00 20 C2 05 20 --03 01 05 40 00 00-- --Set XYZ--

D0 1F 00 -- Set PC to 0x1F00, inside the boot section to enable spm--
64
D2 01 CF 23 movw r24,r30
D2 BF A7 23 out SPMCSR,r26 = 03 = PGERS
D2 95 E8 33 spm
00 <55> 83 <55>

44 - before the first one
And then repeat the following until the page is full.

D0 1F 00 - set PC to bootsection for spm to work
D2 B6 01 23 ll - in r0,DWDR (ll)
D2 B6 11 23 hh - in r1,DWDR (hh)
D2 BF B7 23 - out SPMCSR,r27 = 01 = SPMEN
D2 95 E8 23 - spm
D2 96 32 23 - adiw Z,2

D0 1F 00
D2 01 FC 23 movw r30,r24
D2 BF C7 23 out SPMCSR,r28 = 05 = PGWRT
D2 95 E8 33 spm
00 <55>

D0 1F 00
D2 E1 C1 23 ldi r28,0x11
D2 BF C7 23 out SPMCSR,r28 = 11 = RWWSRE
D2 95 E8 33 spm
00 <55> 83 <55>

Reading Eeprom

66 D0 00 1C D1 00 20 C2 05 20 --01 01 00 00-- --Set YZ--
64 D2 BD F2 23 D2 BD E1 23 D2 BB CF 23 D2 B4 00 23 D2 BE 01 23 xx

66 D0 00 1C D1 00 20 C2 05 20 --01 01 00 00-- --Set YZ--
64
D2 BD F2 23 out EEARH,r31
D2 BD E1 23 out EEARL,r30
D2 BB CF 23 out EECR,r28 = 01 = EERE
D2 B4 00 23 in r0,EEDR
D2 BE 01 23 out DWDR,r0
xx -- Byte from target

Writing Eeprom

66 D0 00 1A D1 00 20 C2 05 20 --04 02 01 01 10 00-- --Set XYZ--
64 D2 BD F2 23 D2 BD E1 23 D2 B6 01 23 xx D2 BC 00 23 D2 BB AF 23 D2 BB BF 23

64
D2 BD F2 23 out EEARH,r31 = 00
D2 BD E1 23 out EEARL,r30 = 10
D2 B6 01 23 xx in r0,DWDR = xx - byte to target
D2 BC 00 23 out EEDR,r0
D2 BB AF 23 out EECR,r26 = 04 = EEMWE
D2 BB BF 23 out EECR,r27 = 02 = EEWE

AVRStudio then reads it back immediately.

An explanation for C2

I think that there are instructions implemented in hardware.

C2 00 / C2 04
ld r,Z+ / st Z+,r
out DWDR,r / in r,DWDR

0: 1001 00Xr rrrr 0001 X=(C2 (bit3))
1: 1011 X11r rrrr 0001 X=!(C2 (bit3)) rrrrr=r16?

These would wrap around. Needs to be tested.

C2 00 --> 0 1 0 1 0 1... (2==0 when address space is only 2 in size)
C2 04 --> 1 0 1 0 1 0...

C2 01 / C2 05
out DWDR,rn / in rn,DWDR

0000: 1011 X11r rrrr 0001 out DWDR,r / in r,DWDR
X=!(C2 (bit3)) rrrrr = (PC & 1F) -- The register number is mapped to the PC.

C2 01 & C2 05 --> 0 0 0 0 0 0...

C2 02

0000: lpm r?,Z+
0001: out DWDR,r?

C2 02 -> 0 1 0 1 0 1...

Created 25 February 2011
By RikusW -- #avr on freenode.net --