ICE-T Command reference manual

Command reference manual for the ICE-T65, ICE-T80 and ICE-6809

Version #1.0

ICE-T was developed and dedicated to “Open Source” by David Banks.
https://github.com/hoglet67/AtomBusMon/wiki

Manual written by A.J. Prosman

August 19, 2019

This page is intentionally left blank

Table of Contents

1. User Manual introduction

The command interface to the emulator is largely the same across the different processors.

For this user manual, we use actual examples for ICE-T65 on an Acorn Atom.

Later sections will show additional commands that are only available on the Z80 and 6809 versions.

2. Features

The emulator provides the following feature set:

- instruction breakpoints (which pause the target processor)
- memory read/write breakpoints (which pause the target processor)
- instruction watches (which are non-intrusive)
- memory read/write watches (which are non-intrusive)
- breakpoints and watches optionally support an address mask, so can cover a block of addresses
- single stepping/tracing every N instructions
- live disassembly when single stepping
- read/write individual target memory locations
- display the processor registers
- hex/ascii dump of a block of host memory
- disassembly of a block of host memory
- calculate a CRC of a block of host memory
- destructive memory test of a block of host memory
- two external trigger inputs, which can be used independently or can be used to make a breakpoint/watch conditional
- a 24-bit processor cycle counter (that is accurate even when single stepping/paused with breakpoints)

3. Getting started

The only client side software that's needed is a simple terminal emulator, connected to the serial port at 57,600 baud.

On Linux, I use gtkterm (for Windows user's Putty or Teraterm can be used) :

```
gtkterm -p /dev/ttyUSB0 -s 57600
```

After powering up the target system, you should see:

```
ICE-T65 In-Circuit Emulator version 0.72
Compiled at 13:40:03 on Nov 15 2015
8 watches/breakpoints implemented
Tracing every 1 instructions while single stepping
CPU free running...
```

The emulator has two modes:

- running mode, where it is executing target processor instructions
- command mode, where it is the target processor is stopped, and the emulator is waiting for command input

The emulator powers up in the running mode, and the target system should run normally.

To change to command mode, just hit return, and you should see the command prompt:

```
Interrupted
09.678286: FE87 : LDA B000
>>
```

The first field (09.678286) is the number of cycles of processor execution (in million cycles). If the target clock speed were 1MHz (which it is the Acorn Atom, this is also the number of seconds).

The second field (FE87) is the address of the current instruction.

The third field (LDA B000) is a disassembly of the current instruction.

At this point, you can list all the command available using the help (h) command.

```
>> help
ICE-T65 In-Circuit Emulator version 0.72
Compiled at 13:40:03 on Nov 15 2015
8 watches/breakpoints implemented
Commands:
    help
    continue
    regs
    dis
    fill
    crc
    mem
    rdm
    wrm
    test
    reset
    step
    trace
    blist
    breakx
    watchx
    breakrm
    watchrm
    breakwm
    watchwm
    clear
    trigger
>>
```

Commands can be abbreviated to their first few characters.

Each of the commands is described in the command reference section later on in this manual.

To see the current value of the target processor registers, use the regs (r) command:

```
>> r
6502 Registers:
  A=01 X=00 Y=09 SP=00F4 PC=FE87
  Status: -V-B----
```

To single step a few instruction, use the step (s) command:

```
s 3
Stepping 3 instructions
13.840846: FE8A : AND #F0
13.840848: FE8C : STA B000
13.840852: FE8F : PLA
>>
```

To change back to running mode, use the continue (c) command.

```
>> c
CPU free running...
```

4 Generic commands

4.1 blist (bl)

List all break points and watch points.

Usage: bl

Example:

```
>> bl
0: 0080 mask FFF0: Mem Wr Watch (trigger: Always)
1: 8000 mask FFFF: Mem Wr Brkpt (trigger: ~T0 and ~T1)
2: B000 mask FFFC: Mem Rd Brkpt (trigger: Always)
3: FF3F mask FFFF: Ex Brkpt (trigger: Always)
```

Information includes:

- a breakpoint number which can be used in the clear command to delete the breakpoint
- the address of the breakpoint (in hex)
- the address mask of the breakpoint (in hex)
- the type(s) of the breakpoint (more below)
- the trigger condition for the breakpoint (more below)

The following types of breakpoint / watchpoint exist:

- Instruction Execute breakpoint
- Memory Read breakpoint
- Memory Write breakpoint
- Instruction Execute watchpoint
- Memory Read watchpoint
- Memory Write watchpoint

Breakpoints cause the emulator to immediately switch back to command mode, and the target processor is paused.

Watchpoints are less intrusive; the emulator remains in running mode, and the processor continues execution without any interruption. An event will be written to the hardware event FIFO, which is then asynchronously logged. The hardware event FIFO is only 512 words deep, and so injudicious use of watchpoints will result in overrun and lost events, as the serial console is much slower.

Multiple types of break point and watch points can be combined on the same address. For example, a memory read watchpoint might be combined with a memory write breakpoint. If there is any ambiguity, then a breakpoint will take precedence over a watch point of the same type.

Breakpoints and Watchpoints can optionally be qualified by a trigger code, which is a function of the two external trigger inputs:

```
Trigger Codes:
    0 = Never
    1 = ~T0 and ~T1
    2 = T0 and ~T1
    3 = ~T1
    4 = ~T0 and T1
    5 = ~T0
    6 = T0 xor T1
    7 = ~T0 or ~T1
    8 = T0 and T1
    9 = T0 xnor T1
    A = T0
    B = T0 or ~T1
    C = T1
    D = ~T0 or T1
    E = T0 or T1
    F = Always
```

This can be manipulated with the trig command.

4.2 breakrm (breakr)

Sets a memory read breakpoint on an address, or range of addresses

Usage: breakr <address> [<address mask> [<trigger condition>]]

This is triggered when the host memory read address ANDed with <address mask> matches <address>, and <trigger condition> is valid.

If <address mask> is omitted, it defaults to FFFF.

If <trigger condition> is omitted if defaults to F (trigger always).

4.3 breakwm (breakw)

Sets a memory write breakpoint on an address, or range of addresses

Usage: breakw <address> [<address mask> [<trigger condition>]]

This is triggered when the host memory write address ANDed with <address mask> matches <address>, and <trigger condition> is valid.

If <address mask> is omitted, it defaults to FFFF.

If <trigger condition> is omitted if defaults to F (trigger always).

4.4 breakx (break)

Sets an instruction execute breakpoint on an address, or range of addresses

Usage: break <address> [<address mask> [<trigger condition>]]

This is triggered when the the address of an opcode fetch ANDed with <address mask> matches <address>, and <trigger condition> is valid.

If <address mask> is omitted, it defaults to FFFF.

If <trigger condition> is omitted if defaults to F (trigger always).

4.5 clear (cl)

Removes a breakpoint or watchpoint

Usage: cl <number> | <address>

The clear command can be used in two ways:

- with a number (e.g. 0-7) it will clear the breakpoint with this number as listed by bl
- with an address (e.g. FF3F) it will clear the breakpoint with this number as listed by bl

When a breakpoint is removed from the middle of the list, the remaining breakpoints are shuffled up so there is no gap.

4.6 continue (c)

Resumes execution of the target code by switching the emulator back to running mode

Usage: c [<reset>]

If is non-zero, then the reset line on the processor will be asserted for approx 100us before execution is resumed.

4.7 crc (crc)

Calculates a CRC over a block of host memory

Usage: crc <start address> <end address>

<start address> and <end address> are inclusive.

The CRC polynomial is the standard "Acorn Atom" CRC-16 polynomial.

Example:

```
>> crc c000 cfff
crc: D67D
>>
```

*D67D is well-known in the Acorn Atom community as the CRC for Atom Basic.

4.8 dis (d)

Disassembles the next 10 instruction in host memory

Usage: d [<start address>]

If the start address is omitted <start address>, the disassembly will continue from the end of the previous disassembly, if there was one, or the current value of the program counter register.

Example:

```
>> d c2b2
C2B2 : LDA #29
C2B4 : STA 12
C2B6 : LDA #0D
C2B8 : LDY 12
C2BA : STY 0E
C2BC : LDY #00
C2BE : STY 0D
C2C0 : STA (0D),Y
C2C2 : LDA #FF
C2C4 : INY
>> d
C2C5 : STA (0D),Y
C2C7 : INY
C2C8 : STY 0D
C2CA : LDA #08
C2CC : STA 0321
C2CF : LDA #3E
C2D1 : CLD
C2D2 : JSR CD0F
C2D5 : LDX #01
C2D7 : STX 06
```

4.9 fill (f)

Fills a block of host memory with a fixed value

Usage: crc <start address> <end address> <value>

<start address> and <end address> are inclusive.

Example:

```
>> f 8000 81ff 01
Wr: 8000 to 81FF = 01
>>
```

*This fills the Atom's screen memory with the character code for 'A'.

4.10 help (h)

This displays the emulator name, version, build date and list of commands

Usage: h

Example:

```
>> h
ICE-T65 In-Circuit Emulator version 0.72
Compiled at 13:40:03 on Nov 15 2015
8 watches/breakpoints implemented
Commands:
    help
    continue
    regs
    dis
    fill
    crc
    mem
    rdm
    wrm
    test
    reset
    step
    trace
    blist
    breakx
    watchx
    breakrm
    watchrm
    breakwm
    watchwm
    clear
    trigger
>>
```

4.11 mem (m)

Dumps a block 256 bytes of host memory in hex and ASCII

Usage: m [<start address>]

If the start address is omitted <start address>, the dump will continue from the end of the previous dump, if there was one, or the current value of the program counter register.

Example:

```
>> m fe00
FE00 71 FE B0 FB A0 10 84 E6 A0 20 20 66 FE B9 00 80  q........  f....
FE10 99 E0 7F C8 D0 F7 20 6B FE B9 00 81 99 E0 80 C8  ...... k........
FE20 D0 F7 A0 1F A9 20 91 DE 88 10 FB 60 69 20 85 DE  ..... .....`i ..
FE30 D0 02 E6 DF 60 88 10 19 A0 1F A5 DE D0 0B A6 DF  ....`...........
FE40 E0 80 D0 05 68 68 4C 65 FD E9 20 85 DE B0 02 C6  ....hhLe.. .....
FE50 DF 60 20 FB FE 08 48 D8 84 E5 86 E4 20 EA FC 68  .` ...H..... ..h
FE60 A6 E4 A4 E5 28 60 2C 02 B0 10 FB 2C 02 B0 30 FB  ....(`,....,..0.
FE70 60 A0 3B 18 A9 20 A2 0A 2C 01 B0 F0 08 EE 00 B0  `.;.. ..,.......
FE80 88 CA D0 F4 4A 08 48 AD 00 B0 29 F0 8D 00 B0 68  ....J.H...)....h
FE90 28 D0 E3 60 08 D8 86 E4 84 E5 2C 02 B0 50 05 20  (..`......,..P.
FEA0 71 FE 90 F6 20 8A FB 20 71 FE B0 FB 20 71 FE B0  q... .. q... q..
FEB0 F6 98 A2 17 20 C5 FE BD E3 FE 85 E2 A9 FD 85 E3  .... ...........
FEC0 98 6C E2 00 CA DD CB FE 90 FA 60 00 08 09 0A 0B  .l........`.....
FED0 0C 0D 0E 0F 1E 7F 00 01 05 06 08 0E 0F 10 11 1C  ................
FEE0 20 21 3B 44 5C 38 62 87 69 40 8D 92 7D 50 DF D2   !;D\8b.i@..}P..
FEF0 9A A2 E2 AE C0 DF D8 D6 C8 C6 C2 48 C9 02 F0 27  ...........H...'
>> m
FF00 C9 03 F0 34 C5 FE F0 2E AD 0C B8 29 0E F0 27 68  ...4.......)..'h
FF10 2C 01 B8 30 FB 8D 01 B8 48 AD 0C B8 29 F0 09 0C  ,..0....H...)...
FF20 8D 0C B8 09 02 D0 0C A9 7F 8D 03 B8 AD 0C B8 29  ...............)
FF30 F0 09 0E 8D 0C B8 68 60 AD 0C B8 29 F0 B0 F4 A2  ......h`...)....
FF40 17 BD 9A FF 9D 04 02 CA 10 F7 9A 8A E8 86 EA 86  ................
FF50 E1 86 E7 A2 33 9D EB 02 CA 10 FA A9 0A 85 FE A9  ....3...........
FF60 8A 8D 03 B0 A9 07 8D 02 B0 2C 01 B0 50 11 AD FE  .........,..P...
FF70 BF 29 10 F0 0A A9 83 8D FE BF 6C FE 7F EA EA 20  .).........l.... 
FF80 D1 F7 06 0C 0F 41 43 4F 52 4E 20 41 54 4F 4D 0A  .....ACORN ATOM.
FF90 0A 0D EA 58 84 FD 4C 00 E0 EA 00 A0 EF F8 52 FE  ...X..L.......R.
FFA0 94 FE 6E F9 E5 FA AC C2 AC C2 EE FB 7C FC 38 FC  ..n.........|.8.
FFB0 78 C2 85 FF 68 48 29 10 D0 06 A5 FF 48 6C 04 02  x...hH).....Hl..
FFC0 A5 FF 28 08 6C 02 02 48 6C 00 02 6C 1A 02 6C 18  ..(.l..Hl..l..l.
FFD0 02 6C 16 02 6C 14 02 6C 12 02 6C 10 02 6C 0E 02  .l..l..l..l..l..
FFE0 6C 0C 02 6C 0A 02 20 E3 FF C9 0D D0 07 A9 0A 20  l..l.. ........
FFF0 F4 FF A9 0D 6C 08 02 6C 06 02 C7 FF 3F FF B2 FF  ....l..l....?...
```

4.12 rdm (rd)

Reads a single host memory address

Usage: rd <address>

Example:

```
>> rd b002
Rd: B002 = F7
>>
```

4.13 regs (r)

Dumps the current values of the processor registers

Usage: r

Example:

```
>> r
6502 Registers:
  A=04 X=01 Y=14 SP=00F6 PC=FE7B
  Status: NV-B----
>>
```

4.14 reset (res)

Resets the processor.

Usage: res

Example:

```
>> res
Resetting CPU
00.000006: FF3F : LDX #17
>>
```

*In this example, you can see the program counter is set to FF3F, which is the 6502 reset vector on the Acorn Atom.

4.15 srec (sr)

Uploads srecords into memory

Usage: sr

There are no paramaters.

You should see a message: Send file now..
Only the S1 srecord is currently understood, e.g.:

```
<S1><Count><Addr><Data>...<Data><CRC>
```

Here's a specific example:

```
S123A0004C10A0A94E8D0802A9A08D09024C33A0A9468D0402A9A08D0502A90F8D04B8A9A9
```

The command completes if no records are received within a short time period.
The linux utility srec_cat can be used to convert an arbitrary file into suitable srecords:

```
% srec_cat somefile -binary -offset 0x1900 -data_only > somefile.srec
```

This will turn a binary file (somefile) into srecord format (solefile.srec), with a loading address in memory of 0x1900.

Note: The srec command was added in the V0.73 elease of the firmware

4.16 step (s)

Single steps one or more instructions

Usage: s [<number of instructions>]

If <number of instructions> is omitted, the last specified value will be reused.

Example:

```
>> s 3
Stepping 3 instructions
00.000008: FF41 : LDA FF9A,X
00.000012: FF44 : STA 0204,X
00.000017: FF47 : DEX
>> s
Stepping 3 instructions
00.000019: FF48 : BPL FF41
00.000022: FF41 : LDA FF9A,X
00.000026: FF44 : STA 0204,X
>>
```

The trace command affects the behavior of step, by allowing the amount of logging to be reduced

4.17 trace (tr)

Controls the frequency of instruction logging produced by the step command

Usage: tr [<number of instructions>]

A value of zero will disable instruction logging.

Example:

```
>> tr 1
Tracing every 1 instructions while single stepping
>> s 10
Stepping 10 instructions
00.000073: FF47 : DEX
00.000075: FF48 : BPL FF41
00.000078: FF41 : LDA FF9A,X
00.000082: FF44 : STA 0204,X
00.000087: FF47 : DEX
00.000089: FF48 : BPL FF41
00.000092: FF41 : LDA FF9A,X
00.000096: FF44 : STA 0204,X
00.000101: FF47 : DEX
00.000103: FF48 : BPL FF41
>> tr 5
Tracing every 5 instructions while single stepping
>> s 10
Stepping 10 instructions
00.000120: FF41 : LDA FF9A,X
00.000138: FF44 : STA 0204,X
>> tr 0
Tracing disabled
>> s 10
Stepping 10 instructions
00.000173: FF48 : BPL FF41
>>
```

4.18 trigger (tri)

Updates the trigger condition associated with the breakpoint/watchpoint

Usage: tr [<breakpoint number> <trigger condition>]

Example:

```
>> break c2b2
Ex Brkpt set at C2B2
>> bl
0: C2B2 mask FFFF: Ex Brkpt (trigger: Always)
>> tri 0 0
>> bl
0: C2B2 mask FFFF: Ex Brkpt (trigger: Never)
>> tri 0 6
>> bl
0: C2B2 mask FFFF: Ex Brkpt (trigger: T0 xor T1)
```

Setting the trigger condition to 0 can be used to temporarily disable a breakpoint.
Omitting all the arguments lists the available trigger conditions.

Example:

```
>> tri
Trigger Codes:
    0 = Never
    1 = ~T0 and ~T1
    2 = T0 and ~T1
    3 = ~T1
    4 = ~T0 and T1
    5 = ~T0
    6 = T0 xor T1
    7 = ~T0 or ~T1
    8 = T0 and T1
    9 = T0 xnor T1
    A = T0
    B = T0 or ~T1
    C = T1
    D = ~T0 or T1
    E = T0 or T1
    F = Always
>>
```

T0 and T1 refer to external trigger input pins on the GODIL.

4.19 watchrm (watchr)

Sets a memory read watchpoint on an address, or range of addresses

Usage: watchr <address> [<address mask> [<trigger condition>]]

This is triggered when the host memory read address ANDed with <address mask> matches <address>, and <trigger condition> is valid.

If <address mask> is omitted, it defaults to FFFF.

If <trigger condition> is omitted if defaults to F (trigger always).

4.20 watchwm (watchw)

Sets a memory write watchpoint on an address, or range of addresses

Usage: watchw <address> [<address mask> [<trigger condition>]]

This is triggered when the host memory write address ANDed with <address mask> matches <address>, and <trigger condition> is valid.

If <address mask> is omitted, it defaults to FFFF.

If <trigger condition> is omitted if defaults to F (trigger always).

4.21 watchx (watch)

Sets a memory write watchpoint on an address, or range of addresses

Usage: watchw <address> [<address mask> [<trigger condition>]]

This is triggered when the host memory write address ANDed with <address mask> matches <address>, and <trigger condition> is valid.

If <address mask> is omitted, it defaults to FFFF.

If <trigger condition> is omitted if defaults to F (trigger always).

4.22 wrm (wr)

Writes a single host memory address

Usage: wr <address> <value>

Example:

```
>> wr b002 e3
Wr: B002 = E3
>>
```

5 Specific commands

5.1 6502 Specific commands

There are no 6502 specific commands

5.2 6809 Specific commands

There are no 6809 specific commands

5.3 Z80 Specific commands

The following additional commands relate to the Z80's seperate IO space commands. There usage is identical to the equivalent memory space commands.

5.3.1 io (io)

Dumps a block 256 bytes of host IO space in hex and ascii

Usage: io [<start address>]

5.3.2 rdi (rdi)

Reads a single host IO address

Usage: rdi <address>

5.3.3 breakri (breakri)

Sets an IO read breakpoint on an address, or range of addresses

Usage: breakri <address> [<address mask> [<trigger condition>]]

5.3.4 breakwi (breakwi)

Sets an IO write breakpoint on an address, or range of addresses

Usage: breakwi <address> [<address mask> [<trigger condition>]]

5.3.5 watchri (watchri)

Sets an IO read watchpoint on an address, or range of addresses

Usage: watchri <address> [<address mask> [<trigger condition>]]

5.3.6 watchwi (watchwi)

Sets an IO write watchpoint on an address, or range of addresses

Usage: watchwi <address> [<address mask> [<trigger condition>]]

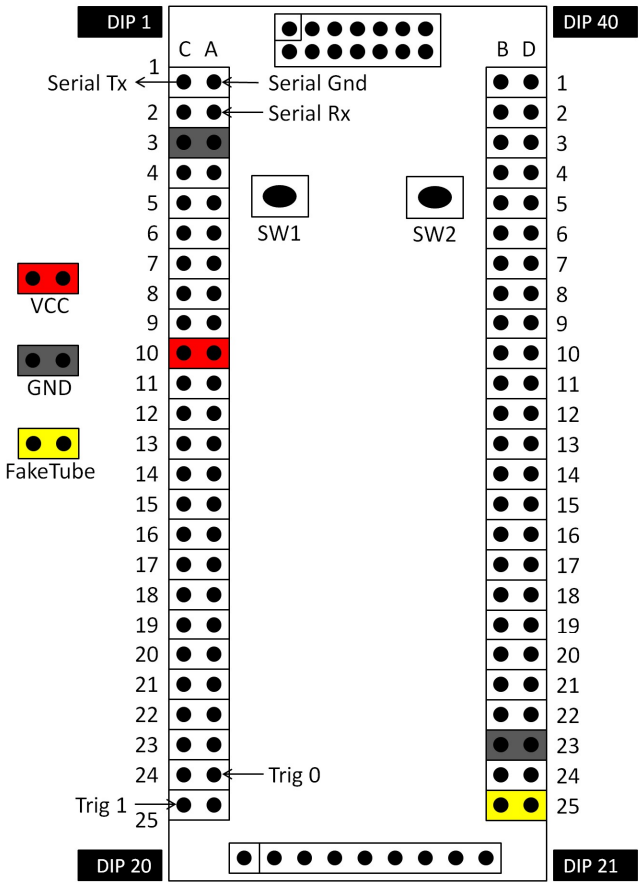
5.3.7 wri (wri)

Writes a single host IO address

Usage: wri <address> <value>

6 Hardware

6.1 6502 jumpers



6502 Jumper Configuration

The two black jumpers connect the GODIL's GND input to DIL pins 1 and 21.

The red jumper connects the GODIL's VCC input to DIL pin 8.

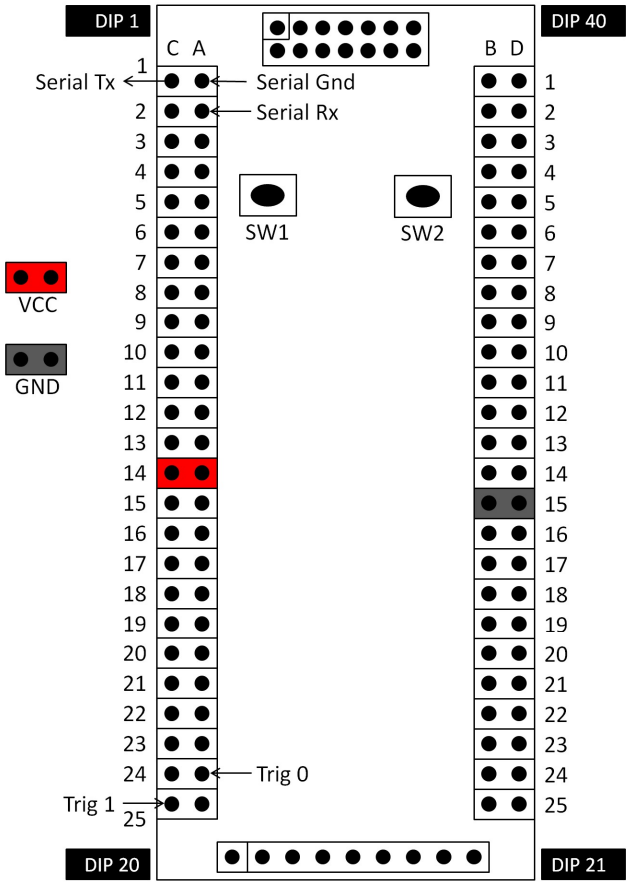
The yellow jumper enables fake_tube mode for the BBC Micro without a second processor.

The pins in the top left corner connect to the serial-USB cable.

- A1 is GND
- C1 is Serial Tx (GODIL -> PC)
- A2 is Serial Rx (PC -> GODIL)

These signals are 5V TTL Levels.

6.2 Z80 jumpers



Z80 Jumper Configuration

The black jumper connects the GODIL's GND input to DIL pin 29.

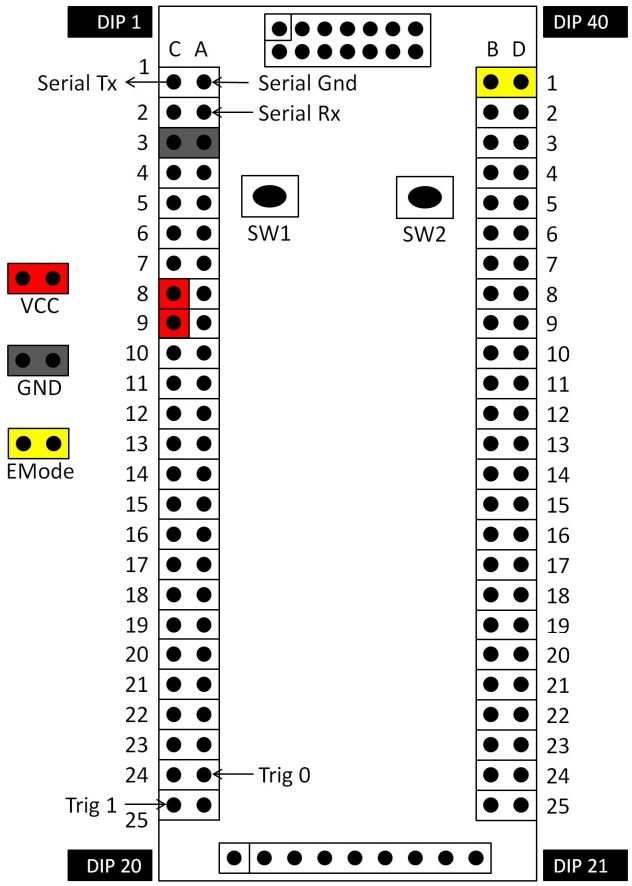
The red jumper connects the GODIL's VCC input to DIL pin 11.

The pins in the top left corner connect to the serial-USB cable.

- A1 is GND
- C1 is Serial Tx (GODIL -> PC)
- A2 is Serial Rx (PC -> GODIL)

These signals are 5V TTL Levels.

6.3 6809 jumpers



6809 Jumper Configuration

The black jumper connects the GODIL's GND input to DIL pin 1.

The red jumper connects the GODIL's VCC input to DIL pin 7.

The yellow jumper enables 6809E Mode, see below.

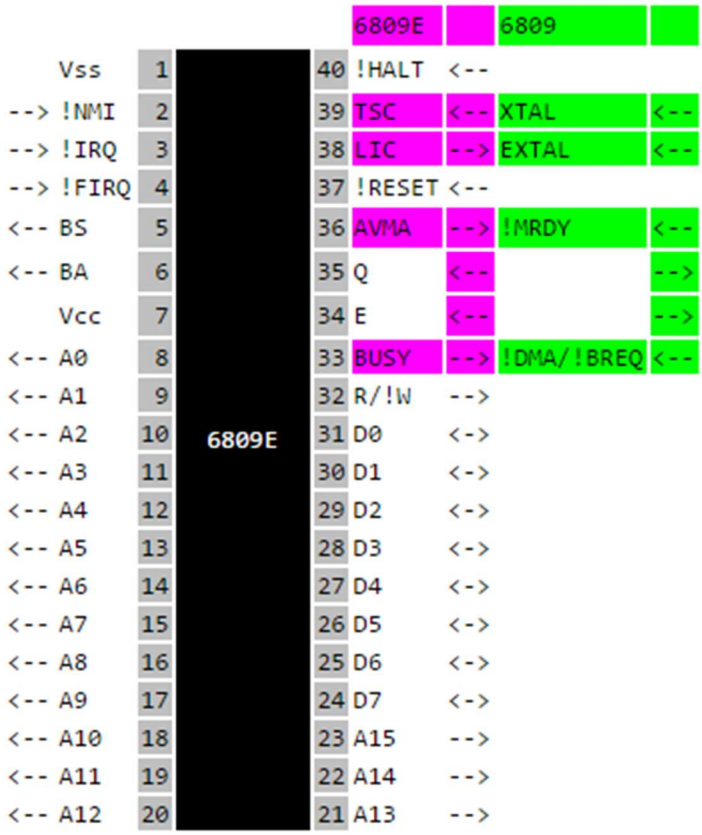
The pins in the top left corner connect to the serial-USB cable.

- A1 is GND
- C1 is Serial Tx (GODIL -> PC)
- A2 is Serial Rx (PC -> GODIL)

These signals are 5V TTL Levels.

6.3.1 6809E Mode jumper

There are some pin-out difference between the 6809 and 6809E processors :



ICE-6809 can emulate either of the processors, and the selection is made via a 6809E Mode jumper shown above:

- Fit the jumper for 6809E mode
- Omit the jumper for 6809 mode

7 Credits and acknowledgements

7.1 6502 FPGA Core

We use the T65 core as a 6502 core. This was started in 2002 by Daniel Wallner. It's currently maintained by Wolfgang and MikeJ from http://www.fpgaarcade.com.

- http://svn.fpgaarcade.com/viewvc/ReplayPub/hw/replay/cores/lib/cpu/t65

The username/password is published on this page: http://svn.fpgaarcade.com.

7.2 AlanD 65C02 Core

We use AlanD's 65C02 core as a 65C02 core. This core is copyright Alan Daly, but permission has been granted for use in open source projects. As far as I know, I have the latest copy, plus some of my own bug fixes:

- https://github.com/hoglet67/AtomBusMon/commits/master/src/AlanD

This core passes Klaus Dormann's 6502 and 65C02 test suites.

7.3 Z80 Core

We use the T80 core as a Z80 core. This was started in 2002 by Daniel Wallner. It's currently maintained by MikeJ from http://www.fpgaarcade.com.

- http://svn.fpgaarcade.com/viewvc/ReplayPub/hw/replay/cores/lib/cpu/t80/

The username/password is published on this page: http://svn.fpgaarcade.com.

7.4 6809 Core

We use the SYS09 core written by John E. Kent, and originally released on Open Cores:

- http://opencores.org/project,system09

The version we are using is 1.26, which is a later version that John worked on for Grant Searle's Multi-Comp project:

- http://searle.hostei.com/grant/Multicomp/index.html
- http://searle.hostei.com/grant/Multicomp/Multicomp.zip

7.5 AVR8 Core

The ICE supervisor software runs on an Arduino-compatible AVR soft core called AVR8,

- http://papilio.cc/index.php?n=Papilio.ArduinoCore

This was originally written by Ruslan Lepetenok, and released on Open Cores:

- http://opencores.org/project,avr_core

The version I'm using is maintained for the open source Papilio FPGA boards by Jack Gassett:

- https://github.com/GadgetFactory/Arduino-Soft-Core