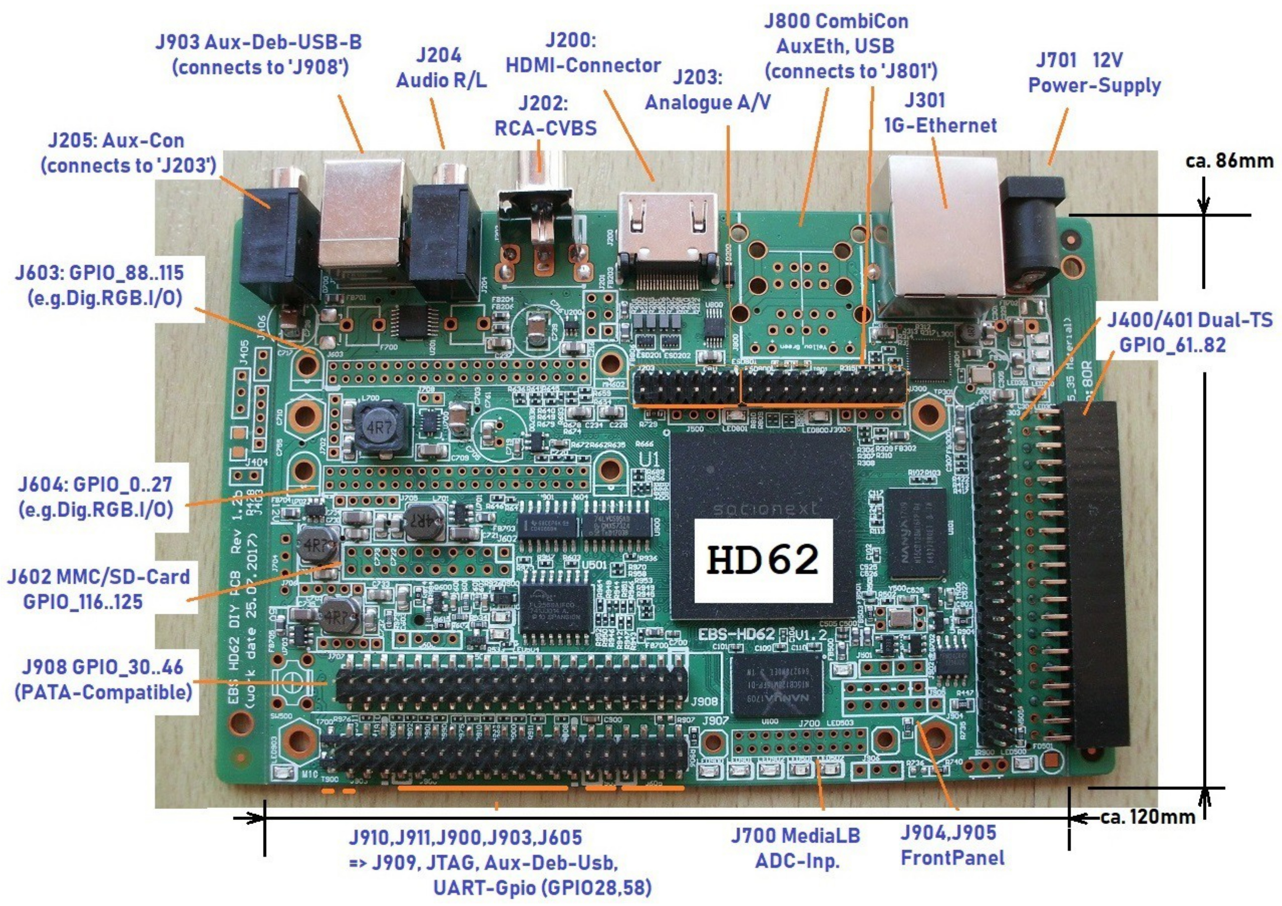
EBS-HD62-DIY-PCB

Public Project-Notes

a 'Multimedia Decoder Board' using the Fujitsu/Socionext MB8AL2030 (HD62) SOC

[EbsHd62Pcb_PubProjNotes_V20200213.pdf]



Picture-1, PCB-Overview with main-connectors, Main-Features

*** EBS-Engineering GmbH product**
- with Socionext MB8AL2030 (HD62) Multi-format HD Decoder

*** Small-Size, can be used as**
- main-PCB for a STB product or other special-applications
- SDK-Platform for own HD62-SW developments
- HW-Reference for own HD62-HW developments

* all HD62-GPIOs and “Special Signals” available via (easy to use) connectors

1.) Introduction & Background

As a background for this description let me first mention a few personal things:
This project description is from 'Eddie (Gerhard Edmund) Bendels'.
I worked many years for Fujitsu-Mikroelektronik in Germany before leaving in 2014
when I started my own company "EBS-Engineering GmbH" (www.Ebs-GmbH.net).
Note that this text is written in "I" perspective but is equivalent from my company view.
(If you are curious, you will find more personal information under this link
 [TODO ______________________])

Fujitsu-Mikroelektronik was the European subsidiary of Fujitsu-Limited in Japan.
Over the years I worked there in different departments involved with various products.
Describing all things here in detail would lead to far, but 'typical application-engineering work'
(or playing with different toys) describes it best.

In 2015 Fujitsu went trough a restructure and actually evolved into a new companies.
Basically the 'Multimedia-IC-Department' which I worked for the recent years
belongs to the new company Socionext now.
The last project I was involved with, was a multimedia-decoder-chip/SOC development.
Meanwhile the chip is sold under the product-name 'MB8AL2030'
but the internal project name was 'HD62' which I still prefer to use here.
Note: in the following I will simply use “SN” to refer to Fujitsu/Socionext.

When starting my own company activities, I decided I want to have 'a decoder-board-product'.
After few considerations I decided to choose this 'HD62' as the main-component for it.
The main reason is obvious because of my experience with this chip (HW and SW)
and still good connections to the people at SN.
So what I developed and describe here is basically a 'Multimedia Decoder Board' using the latest
decoder-chip from that time, which I named 'EBS HD62 DIY PCB'.

The 'EBS HD62 DIY PCB' can simply be used “where it fits in“. This could be
- as a (little) sub-PCB-component in more complex systems
- as a SDK-Platform for own HD62-SW developments (alternative to the SN development board)
- as a HW-Reference for own HD62-HW developments
- for a STB-product, when the PCB is mounted into a suitable housing (with a FE-PCB)
(Actually I had the plan to offer such STB under the the product-name 'EBS HD62 DIY STB'
 but I never found the time to bring this into “ready state” for the end-customer market)

As a design house, also offering PCB-development services, I can also use this development
 as a basis for customized PCB-designs, where parts of the schematic and layout can be re-used.

Since I put a lot of efforts in the PCB development, I was initially concerned that part of my work
could easily get copied. So I kept information about my design and product confidential
and provided this only to specific business partners and customers under NDA.

Meanwhile a lot of time has passed and the HD62 is 'kind of old' now.
There are many similar and more powerful SOCs around today
and a lot of information is freely available which I find very inspiring.
Thus I changed my mind and decided to created this
 “EBS-HD62-DIY-PCB Public-Project-Notes” document for the public now.

It also allows me to simply refer to it and easily provide interested people some impression about my past working background and area of experience.

I do not plan to put further efforts in actively promoting this product,
but the 'EBS HD62 DIY PCB' can still be ordered by business to business customers (B2B).

Note that surprisingly the 'MB8AL2030' is sometimes even still used for new designs
for specific reasons which I mention in Appendix-A.

Actually I also imagine that I can use this PCB for specific things in the future,
like for realizing "special test equipment".
A list of interesting chip-features for that is mentioned in Appendix-B.

2) Key considerations:

During my 'Fujitsu times' I had many close contacts to customers designing and manufacturing STB products. I was always a bit sad, that they developed their PCBs strictly and just to their product-requirements.

I would have loved to purchase their final STB-products and 'take out the PCB' to be able to (miss-) use it also for kind of hobby applications. Generally purchasing such STBs would have been a nice solution to get hold of 'cheap processor/decoder-boards', if these would also allow to easily connect-up own HW extensions to it.
However on these customer designs, e.g. 'unused I/O-interface signals' are usually 'not connected up'. Leading these signals to 'optional connectors' was considered to much extra work and the extra PCB-area might have cost extra.
Looking back now, considering all these 'Raspberry Pi® type DIY boards' on the market today, I should have pushed this idea more forward ...

Now, when starting to design my 'EBS HD62 DIY PCB' (back in 2015), my key-considerations/targets were:

2.1) I want 'all I/O-features' of the HD62 available, either on 'outside-connectors' or on (standard, simple to use) 'pin-headers' for 'plug-on extension PCBs' (or 'shields' as they are called today in the Pi and Arduino® world).

2.2) I want 'outside-connectors' just on 'the rear side', so if the PCB is mounted inside a 'typical STB case', all connection-cables are only at the back. ('some DIY boards' have cable-connectors 'all around', which makes PCB-layout easier, but it's kind of unhandy on the desk.)

2.3) I want to keep it 'as simple to use' and 'as low cost' as possible choosing only 'commonly found' peripheral components and connectors and choosing 'just a 4-layer PCB stackup'.

2.4) I want the possibility to easily measure certain things on this HW.
For example, the current in various supply voltages or some other interesting signals.
(On 'typical customer boards' which I worked-on in the past this was often very difficult.)
So in my design, I added shunt-resistors for current measurements,
extra 'foot-prints' for pin-headers (which are usually not populated/mounted),
and some 'test points' (which can be also resistor-pads and 'vias').

2.5) I 'do not want' that implementing consideration 2.4) makes my board as huge as typical 'Evaluation' or 'Development-Kit' boards.
So I spent a massive amount of time to find an 'optimal very dense component placement'.

This required a lot of iterations, where I placed components very narrow and tried to route it.
Often, after hours of trying, I found out it's not possible and then had to un-route certain things again, place components 'a millimeter further apart' and try again.
Finally I am proud (at least believing) I achieved a component-placement and routing as highly optimized as possible :)

3) Comments on "Schematic Pages/Sheets"

The 'EBS HD62 DIY PCB' was developed using the Altium® software
and the schematic pages can be found in the “EbsHd62Diy_Sch_?_Vxxxxyyzz.pdf” outputs.

NOTE: there is no 'PCB-layer-print' included, since I prefer to keep details on the PCB-layout closed at this point. If you are interested please get in contact.

3.1) 'EBS_HD62_PCB_Notes.SchDoc'

The schematic-pdf is available in two print-out versions.
- 'EbsHd62Diy_Sch_c_Vxxxxyyzz.pdf' is a 'color-version' including pictures, which is a bit nicer to look at on a PC-screen.
- 'EbsHd62Diy_Sch_m_Vxxxxyyzz.pdf' is a 'B&W-version' without pictures, more suitable for printing out.

The smaller-picture on the 'color-version' shows the 'EBS HD62 DIY PCB'
with a "JTAG-Debug & RS232 Adapter-PCB" (EBS-Hd62-SimpleDebugInterface) plugged-on.
Details on this are mentioned in appendix-C.

A note on the 'schematic style': Often if PCB-designs consist of multiple schematic pages/sheets, there exists a 'Top-Sheet' which connects up local signals/net- labels.
You also find that connections between pins/signals are all connected-up by lines.
This makes it easy to follow 'all signal connections' but takes a lot of 'paper space'.
Sometimes there are so many connection lines that things get actually difficult to follow.

So 'my preferred schematic style' is to make extensive use of 'net labels' which are 'all global' in this design. This is because I prefer 'dense schematics' which fit also onto 'printed A4-pages'.
I regard this design as 'not complex enough' for needing a top-sheet & local symbols.
If you have difficulties to find out about the 'pin/signal-connections', simply use the 'search text feature' of your PDF-Viewer to find them.

3.2) 'Hd62Main.SchDoc'

The HD62-chip 'U1' appears as different symbol-blocks 'U1A .. U1L' (x of 22) on the various pages.
On the main page, we have symbol-blocks with GND & power-supply-pins, control signals like Reset, Clock-Generation and so on connected up.
Generally note that the HD62 has one (most effective) power-saving option where 'most supply voltages' can be switched off and only the so called 'power-island-block' remains under power.
Note that supply-voltages which remain 'always on' end with 'xxxAO',
the others (usually) with 'xxxSW'.

So, other supply voltages on the board can be effectively 'switched On/Off' controlled by the 'HD62_PUP' signal which corresponds to a 'control-register-bit' accessible by SW.
Obviously when powering-up the board (e.g. when the ext. 12V board-supply-voltage is attached) this signal becomes active, so the board and SW can boot up.

For this, the '#MST_RST' signal is relevant, which causes the 'HD62_PUP' to get 'active high'.
If the Application/Control-SW decides to activate the 'low-power-mode', the 'HD62_PUP' signal will become 'inactive low' and most of the board supply voltages go off, also shutting down the processors, so there is no more SW running !

Now there are a few options how the HD62 can 'come out of' this mode again which is basically
- triggered by a 'timer which was programmed accordingly before', or
- a specific 'Infra Red-Key' detected, (IR-Input logic) so e.g. triggered from a remote control, or
- triggered from the 'CEC signal' from the HDMI-connection.

Note that 'coming out of this standby-mode' causes the '#SYS_RST' to generate a reset-pulse,
and that the SW will 'boot up again from the beginning',
so this is not 'a simple sleep mode' where the SW can simply continue where it stopped before.)

That was a lot of explanation for this low-power mode option.
In practice, there are also applications where the complete "HD62-PCB Supply"
is simply switched On/Off by the 'main system' (e.g. if it is just a sub-PCB component).

Or, as another example by a 'low-power standby MCU' (e.g. on a STB-front-panel)
which would have the advantage that it's SW can still do some 'intelligent work'
while the HD62 is simply switched-off.
For this case, there are some 'resistor configuration options' R736/R740 which would allow such
MCU to effectively control the HD62_PUP / 'EN_POW' signal instead.

We also find the U501 "S-Flash-Memory" (SFlash) on this sheet.

Note that after a '#SYS_RST', the HD62/CPU starts executing from a 'small on-chip-boot-ROM'.
This boot-code will check 'logic-levels' of some (during boot-time) dedicated GPIOs (GPIO33..38).
On this "HD62-PCB" these are set by pull-up/-down resistors such that
the boot-code (will try to) install the "application-SW" by copying sections from ext. SFlash
(attached to GPIO55,83..86) into the external DDR3-memory and then 'jump there'.
For more details on the boot-process see appendix-D.

3.3) 'Ddr3Mem.SchDoc'

This page shows the connection to the 2x DDR3 (DRAM) memory chips.
Note that the HD62 has 16-bit-data-buses to each DRAM, and can address only max. 4GBit
memory-chips. (2x 4GBit => total 1 GByte)
Note: There exists a '_A15' address signal which would allow to address even bigger memory-chips
but due to the implemented 'memory-map' this is basically not possible,
so the '_A15' signals are simply not connected up here.

3.4) 'AvIo.SchDoc'

The circuits on this page contain the logic for analogue and digital AV-signals.
- Basically 'digital AV' is available via the external 'HDMI-connector' (J200)
- (some) 'analogue' signals are available via the external 'RCA/Chinch- and Audio-Jack'
  (J202/J204) connector.

All analogue AV-Signals including RGB are also available on the (internal) pin-header 'J203'
e.g. for plug-on ext. boards.
Note that there exists a 2nd ext. 'Audio-Jack connector' (J205) leading to the pin-header 'J201'.
When using 'J201' as a 'pin-header' this provides an option to bring the 'outside signals 'AUXIN1/2'
basically to the 'inside'.
When using 'J201' for 'putting on jumpers', the GPIO127-signal (which could be configured for
SPDIF-output) and the '3V_AUX' supply could be “brought outside”,
thus a little coax- or optical- SPDIF-output adapter solution could be attached.

3.5) 'TsInpEtc.SchDoc'

On typical STBs (not like just 'media player' solutions), the 'media-content' is usually coming from satellite-, cable- or terrestrial front-end [FE] demodulation-circuits.
On the "HD62-PCB", such front-end (extension-PCB) could be connected up via the J401/J400 connectors/pin-headers.
These provide the necessary signals (GPIO-pins) for connecting up 'two Transport Streams' [TS] for a 'dual front-end'. Usually the 'J400 angle-connector' should provide the most practical mechanical solution to connect up a FE-PCB. (horizontal side attached)
Since J401 has an identical pin assignment, the TS-signals can also be passed to a 'plug on ext. board', which would be interesting e.g. for a 'Common-Interface' [CI] plug-on board.

3.6) 'GpioDef.SchDoc'

On this page, we have the HD62-symbol-blocks with it's general purpose (GPIO)-signals and respective net names. In addition to 'general purpose functions' which are fully SW-configurable, some GPIO can be configured to have a 'specific function'.
For such usage, they are connected up here to respective pin-headers where appropriate ext.boards could be plugged on.
(actually also an SD-Card-Connector (J601) could be mounted/soldered at the PCB-bottom)
Also on this page is the 'J605'-pin-header/connector-symbol,
where a pin-assignment compatible to the "ESP8266-Module" was chosen.
Note that 'J605' is a special 'on top of footprint of' J909' case, described further under 'UpiGpio.SchDoc'

3.7) 'Ethernet.SchDoc'

This page contains all components for the implemented Gigabit-Ethernet-Interface.
A few unusual things to mention here are:
Usually there are 'just status LEDs' in the RJ45 transformer/connector, but since this connector is usually located 'at the back' it's difficult to see what the LEDs are doing
(unless one looks 'from behind').
So I put extra LEDs on the PCB, so these are easy to view from 'the top or also from the front'.
Another interesting 'measurement option' are the signals '_EXE' and '_REV' described in another document. ([ToDo: Link to AppNote measuring actual/effective Ethernet data rate])

3.8) 'UsbInterface.SchDoc'

This page contains the interface-circuit and power-switch for the two USB-(2.0) ports.
The USB-Signals are available on the pin-header J801 for 'internal plug-on board' usage.
In addition we have two options regarding external connectors:
a) just a "dual USB-(A-Type) connector" could be populated to have external USB-connectors.
b) a "combined RJ45 / dual USB -connector" could be populated instead.
In addition to the the 2xUSB-connectors, we will then have the
“additional-RJ45 connector-signals" connected to the (internal) pin-header J801,
so this would allow to plug-on an 'internal plug-on-board' which then has access
to the 'external world' via the "RJ45" pins. This could be used to realize a 2nd Ethernet-Interface.

Note: The reason for this option a) or b) is that "combined RJ45/dual USB-connectors" are quite high, so actually no connector is populated at manufacturing. The desired connector can easily by 'soldered in', depending what is needed.

3.9) 'UpiGpio.SchDoc'

This page contains further Gpio-Connectors and some 'trick circuit'.
To begin with, we have the J908 and J907 pin-header-connector called "UPI-Bus" on an "P-ATA compatible connector". Note that these connectors are placed simply 'on top of each other' (in the schematic and also on the PCB).
The reason is simply that I wanted to have 2-different foot-print options on the PCB.
One is a pin-header as 'SMD connector' and the other as a (through-hole) 'TH-connector'.
(The background is, that the J601-SD-Card-Connector at the bottom can only be mounted if 'SMD-pin-headers' are used at the top !)

"UPI-Bus" stands for Universal Processor-Interface. Things like 'Parallel-Flash' or the 'CPU-Interface' for a "Common Interface Adapter PCB" could be connected up here.
Also 'old P-ATA hard-disks' have a kind of compatible 'CPU-Interface' and the HD62 has logic to drive it. So the pin-assignment was chosen such that a 'PATA->SATA' conversion/adapter-PCBs could be mounted here as well.

Note that similarly "on top of" J909 (which is a 2x20 'SMD connector' footprint), other 'TH-connectors' are placed (arranged next to each other as 2x20-pins in total).
These are

- jumper J910-1	/J909-3	making available the 'JP_FRES' signal.
- jumper J911-1	/J909-1	UPI_A5 or 5V0SW
- jumper J911-2	/J909-2	OPT5V
-	/J909-5,6	not used (could be 'cut out')
- header J900-1-20	/J909-7-26	a 'standard 20-pin JTAG-/debug-connector'
-	/J909-27,28	not used (could be 'cut out')
- header J903-1- 4	/J909-29-32	just connecting to the J903 USB-(B)-Connector
- header J605-1- 8	/J909-33-40	containing Rx,Tx I/O-pins e.g. for a RS232-Interface (or ESP8266-Module)

Note that when the "JTAG-Debug & RS232 Adapter-PCB" is used, it 'plugs onto' pins J909-7-40 !

On this page we also find U901, a (old fashioned) quad Mux-Switch, and U900 a 74LCX595 serial to parallel-shift-register.
This is a little 'trick-circuit' is used to realize some additional "control-output-signals" without having to use-up any of the 'free' HD62 GPIOs for this.
(Which would then not be available for extension anymore).

What I use here as a 'mux-control signal' is the SFlash-Chip-select signal (IO55).
If the HD62 accesses the ext. SFlash signals (IO83..87), the shift-register-signals are 'pulled' to some inactive/default levels. If the HD62 does not access the ext.SFlash, IO83..87 can be configured by SW to 'bit-bang out' data into the shift-register. (Note that C906 is necessary to fix some 'timing-glitch' when the 'SFlash-Chip-select signal' becomes inactive)
The shift-register outputs are connected to "User LEDs" and for other purposes.

3.10) 'PowerSupp.SchDoc'

This page contains mainly "switch mode voltage regulators" for the various supply voltages.
Note that U700 is the main-regulator providing a 5V-supply which is 'always' on.
The other 'SY800-regulators' can be 'switched Off/On' from the 'EN_POW' signal.
Also some 'linear regulators' are used here to provide voltages for the 'HD62 power-island' (REG701=>3V3AO, REG702=>1V2AO) (which must provide just 'few mA').

6) Appendix-A, Considerations if using the HD62 still makes sense for new designs

6.1) A simple criteria (and probably the 'most missing feature') is that theHD62 can only decode and output up to "full HD" content (1920 x 1080, max 30p),
so if your application really needs the new "4K" resolution, it's a knock-out criteria.

If "full HD" is fine, then it's basically a question of:
- does your application need the 'top features' of 'modern decoder-chips'
 (like 'fast CPU-clock frequency', lager DDR Memory support, latest GPU features, etc)
 or is the main-application simply “be able to decode MPEG2/H.264/H.265 content”.

- does your application need a 'most recent' (Linux / Android) SW-environment with standard
 SW-interfaces, where a “standard application” can easily be ported onto all platforms,

- or, is your application “kind of special anyhow” where the SN ported Linux
 and their own (Fapi) driver-interface for “media decode-functions” can be used.

- a huge plus is also, that most of the “fapi-drivers” are provided in “source-code”
and that a device-manual with register-description can be made available (under NDA).

6.3) if you have a kind of “special application”, where a complex Linux SW-environment
 is not even necessary and just “some HW features” are effectively used,
 then using the EBS “APSOL” SW-basis might be an interesting alternative.

The “APSOL”-SW stands for 'Advanced Primitive (SW) Solutions' and provides
a 'simple / minimalist' SW-environment with drivers and application examples
for easily realizing kind of 'primitive super-loop' application-programs.

Actually 'the catch' is that this “APSOL”-SW environment is supposed to easily
compile and run also on various other embedded-system and MCU platforms.
Even allowing to kind of simulate/model/test such applications as PC-Programs
on Windows- or Linux-PCs is possible. (see [ToDo_Link] for more details on this)

6.4) Probably still 'a unique pro-features' of the HD62 is also:
- there exists a specific HD62-part-number with “automotive specification”
  (other decoder-chips are usually only available 'with consumer-specification')
- since the “HD62” is an “older (not so powerful) device”,
  it might also offer lower power-consumption (about 2W) compared to other solutions

6.5) Regarding “sourcing” it is also possible to purchase the HD62 for 'smaller companies/projects'
  (admittingly the price and lead-time might not be so good)

On the other side, consider that for some of the 'modern decoder chips',
it might be possible to purchase 'cheap DIY-boards' but actually difficult to purchase ICs
for own PCB productions.

Up do now, Fujitsu/Socionext was also very good regarding 'long time availability' (EOL).
'Modern decoder-chips' might be available for only a shorter time-frame until new chips show up.

7) Appendix-B, ideas using the 'EBS HD62 DIY PCB' for special applications

The HD62 has interesting HW features, so many things which could be done with it are imaginable. However since nowadays also other platforms like the "Raspberry-Pi 4" are available, it's always a question of which platform is better to use, and meanwhile often others might be the better choice.

So in the following I mention only a things which I imagine are 'still interesting enough' (feasible) to be done using the HD62 or the 'EBS HD62 DIY PCB'

7.1) A simple (also H.265/AVC) "Media Decoder" board
- actually this is the application of my current 'EBS HD62 DIY PCB' customers.
Based on the "Fujitsu AV-Decode" I have developed a "EBS.LAV-Decode" application with a self defined "SW-control protocol", operated via a "2nd UART-Interface" using free GPIO-pins.
It allows another "main-processor system" to control it.
The control protocol has e.g. commands for acquisition of SI-Information present in the TS-Input, and to start decoding a specified service.
Here the HD62 also has the advantage that is has a "TS-Input" (with DVB-descrambler options) which might be an issue for other 'just media decoder' chips.

7.2) Showing "Media Content" on connected "RGB-Displays"
- the HD62 has may GPIO-pins, some can be configured as "dedicated RGB-output"
  allowing to drive up to 2x TFT/LCD-displays directly connected to a
  3x8-Bit-Data-Bus plus H/V/EN signals. (other solutions might need interface ICs for that)

7.3) Capture the content of some other "Media-Systems" output

- instead of "RGB-output", these GPIOs can be also configured as "capture input".
 In this mode, various options like 3x 8Bit, or 16/8-Bit CCIR are possible.
 (actually a 1st prototype of a "plug-on HDMI-capture PCB" already exists)

- this approach could also be used to capture the "DVI-output"
 (plus emulated Keyboard/Mouse controls via USB or PS2) of a PC
 for "remote desktop" applications, thus not depending on a "VNC-server-application".

The captured content will then basically be stored "picture by picture" in associated memory-buffers, and could be encoded using the on-chip H.264 encoder,
and then e.g. "streamed out" via Ethernet.

- Since there exist also a "binary capture mode" (and the possible data-rate is relatively high) one could imagine to use this capture feature (with an additional CPLD or FPGA for 'trigger control logic') to realize a 'logic-analyzer' or 'digital storage scope'.
Such implementation would have "a large capture buffer" using the available DRAM.

7.4) Reception of a “Life-Service” and “Streaming out the Trans-coded Service”

Usually a STB is used to “watch” a Satellite- (or Cable- or Terrestrial-) Channel on a TV.
(Nowadays LCD-TVs basically have a STB functionality included, so this market is shrinking.)
A STB based on the HD62 could use it's decoder- and encoder-feature to
“stream out” e.g. a re-formatted or down-scaled and thus bit-rate reduced version
via “the network”, maybe event to your “vacation location” in a different country.
This application is not new, but maybe interesting when existing solutions might become
unavailable in the future.

7.5) Ethernet “TAP” Monitor

The HD62 has a 1GBit Ethernet-Interface. With a 2nd “Ethernet extension PCB”
(where also a first prototype exists) the 'EBS HD62 DIY PCB' has two Ethernet-ports.
This could be used to “insert” it into an 'Ethernet-Cable connection' of another system,
and with an appropriate SW it could pass the Ethernet-traffic between these ports,
also monitor/analyze the traffic, and manipulate certain packets for special tests.
Obviously other solutions (e.g. using a PC with multiple Ethernet-cards) are possible,
but 'for quick hacks and implementing special tests' using my “APSOL” SW-base
this is very easy to realize, at least for me :)
[ToDo: Link to AppNote]

7.6) More ideas might be added in future ...

8) Appendix-C, the JTAG-Debug-Adapter (and RS232 Interface)

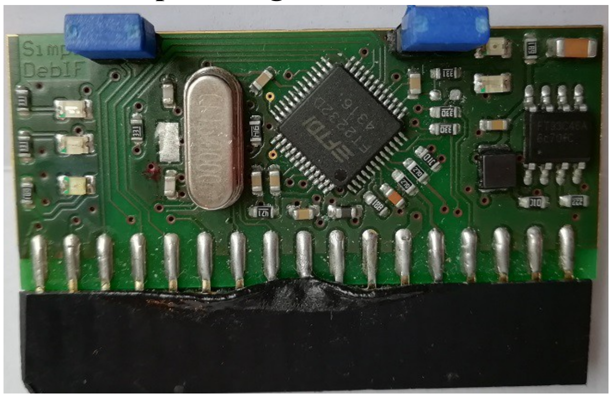
The 'EBS HD62 DIY PCB' basically provides a 'standard 2x10-pin JTAG pin-header' and a pin-header with Rx/Tx-UART-GPIOs where a “RS232 level shift adapter” could be connected.

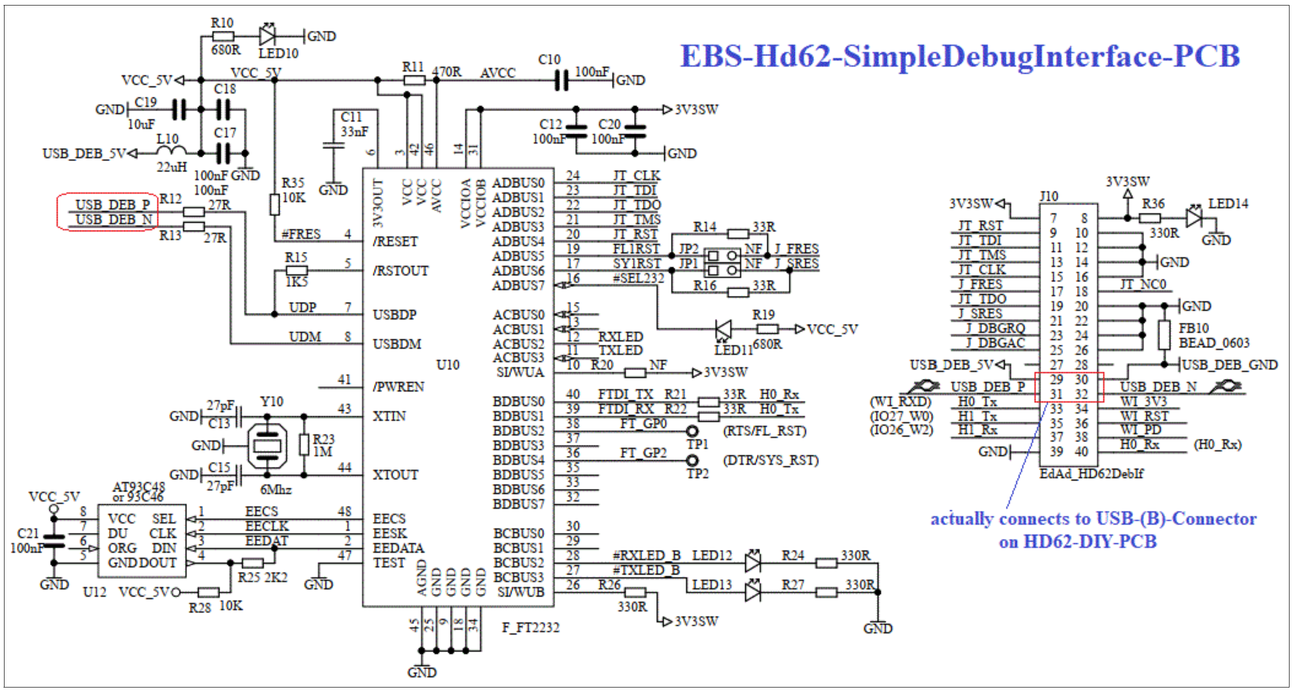
Since these pins are 'combined' on the 2x20-pin header J909 (described in 'UpiGpio.SchDoc') actually pins 5/6 and 27/28 would have to be 'cut-out' to be able to plug on a 'standard JTAG' connector. (The Fujitsu/Socionext SW supports 'JTAG- debuggers' from Lauterbach and ARM.)

Alternatively, the Fujitsu/Socionext “black box debug adapter” could be used. It contains the commonly used FT2232 (Dual USB to Serial UART/FIFO IC) chip for such purpose.
A “Download-Utility” from Fujitsu/Socionext can be used (using the FT2232-Port-A) to 'download & run' programs to the HD62 target-system. (In theory, this JTAG interface could also be used with the “GNU-Debugger” but I am not certain if this is option is practically usable)

The FT2232-Port-B is usually used as a simple RS232 to USB-bridge allowing to connect to a 'PC-Terminal-program', used for HD62-application-program print-outs or as simple user-interface.

As an alternative to the above, I developed the **“EBS-Hd62-SimpleDebugInterface-PCB”**.
It is HW compatible with the Fujitsu/Socionext “black box debug adapter”, but fits directly on the J909 2x20-pin header. (act. pins 7..40 = 2x17 pins)
The “nice thing” with this solution is that the actual USB-(B-Type) connector for the PC-debug-connection is not on this “SimpleDebugInterface” but uses the one on the 'EBS HD62 DIY PCB', so the USB-bus is actually also connected via J909 as well.
NOTE: Since the FT2232 is 'relatively expensive' it is also possible to use/build a similar/alternative PCB-solution with just a “USB to RS232-bridge” like the “CH340 or PL2303" for only RS232 e.g. when the 'JTAG download & run' is not needed and the "Ebs.EBoot-App" used instead.





9) Appendix-D, Boot-Process (from S-Flash)

As mentioned before, 'in the final application' the on-chip boot-monitor
will try install the (application) SW from SFlash into the DRAM-memory and start it.

If the SFlash is 'still empty' (no program-code and header-info stored yet) the 'on-chip-boot SW' will simply remain running in a 'polling-loop'. In such case, it is also possile to 'download' the "application-SW" via a "JTAG-Debug-Interface" into the DDR3-memory and start it.

Actually this method can also be used to "program something into an empty SFlash"
by downloading and starting a special "SFlash Programming Application/Utility".

This 'just download & run method' can also be used during the "Application-SW-Development" where always storing the App-SW into the SFlash first, is simply to time-consuming.

However his "download & run via a JTAG" gets also time consuming for 'bigger code-sizes' like Linux-Applications !
For such cases, I have developed (and use myself) a different approach.
I have a 'special secondary boot-monitor app' starting from SFlash called "Ebs.EBoot",
which then supports to download the 'final application-code' via the Gigabit-Ethernet-Interface.
(It uses a “self defined Ethernet protocol” and corresponding Ethernet-Monitor PC-Uitility called “Ebs.EMon”. I believe this own solution is much easier to deal with than e.g. “UBoot”)

If the "Ebs.EBoot" detects, it is not used for "Ethernet-download" it will 'install/copy'
the 'Final (Linux) Application' (e.g. the media-decoding application "EBS.LAV-Decode")
from a specific SFlash-area into memory and start it.

For further info on the "Ebs.EBoot" and "EBS.LAV-Decode"
see 'Todo: Link2 Ebs.EBoot/EMon and EBS.LAV-Decode' info.

10) Special thanks

There are many persons and companies who supported my work so at this occasion I like to thank

- (Ex-) Colleagues from Fujitsu/Socionext in Europe, Japan and China

- **www.sr-systems.de** and **www.rsdtv.com**
  for guidance and help on my 1st PCB-prototype production

- **www.we-online.de**
  for providing various samples for 1st prototype production

- **www.yinhe.com** who I could gain as the manufacturing sub-contractor for me.
  Very special thanks to Eric Tian for his patience and efforts
  to manage things from component purchase, production of the HD62-PCBs,
  and all the commercial and export aspects.

- and thanks to my clients for purchasing my 'EBS HD62 DIY PCB' product

Revison History

2020/02/13 'temp version 4 review ' (before public release !)

Contact-Info:
Edmund Bendels
Am Forst 28
64807 Dieburg, GERMANY
Phone: 0049 6071 21817
Mobile:0049 1637327540
EMail: info@EBS-GmbH.net