Reverse Engineering of advanced RISC-V MCU with USB3 & High Speed peripherals

GreHack 2022

Who ?

Benjamin VERNOUX @bvernoux

Embedded Hardware / Firmware / Host tools

SDR: AirSpy R0-R2/Mini

HydraBus v1 / HydraNFC v1&v2…

HydraUSB3 v1

HydraSCA-LISN v1

Why this Research / Reverse Engineering Project ?

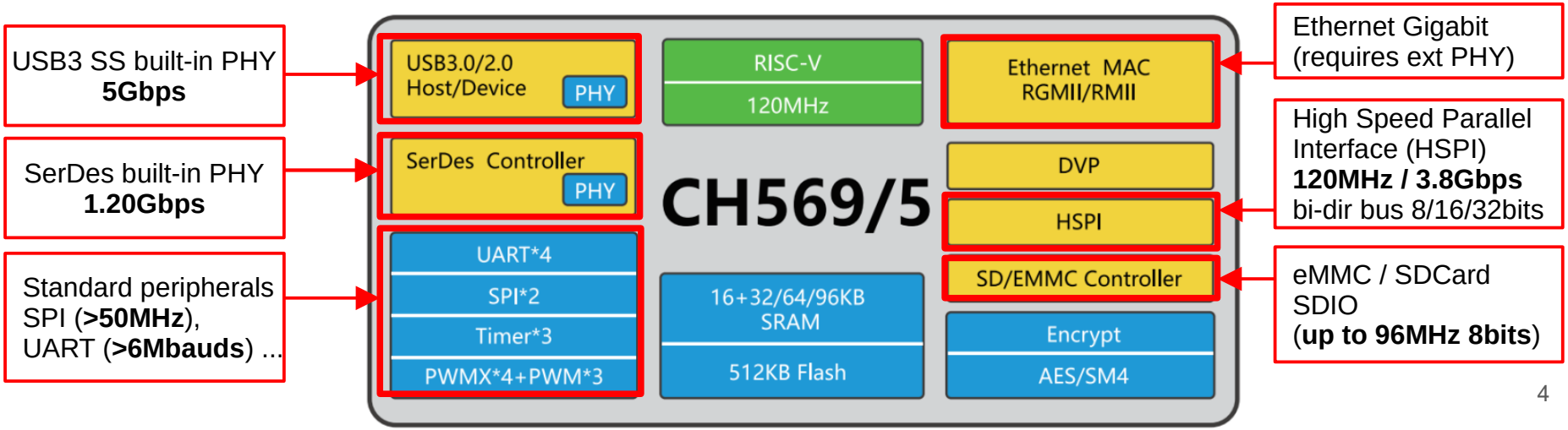
- **Chip Shortage** “everywhere” …
- Find something “**available**” with “RISC-V” with interesting features

WCH CH569 MCU details

Why WCH CH569 ? => Cheap / Available / Amazing features (for “Excellence and Fun”)

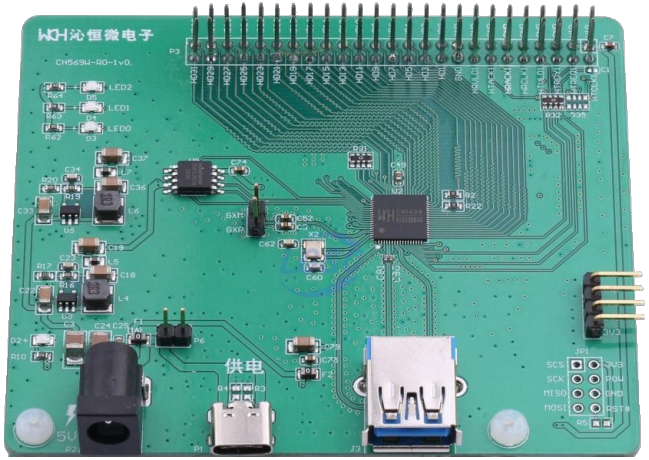
RISC-V 32-bit MCU @120MHz
WCH Proprietary “RISC-V3A” architecture see CH32V103 datasheet (which is based also on RISC-V3A with more details)
RISC-V core **RV32IMAC** 32bits Base **I**nteger instruction + Integer **M**ultiplication/Division + **A**tomic + **C**ompressed instruction

- **Fast Interrupt** Automatic save/restore of registers, use non standard **attribute((interrupt(“WCH-Interrupt-fast”)))**
- **64 bits SysTick** (cycle accurate)



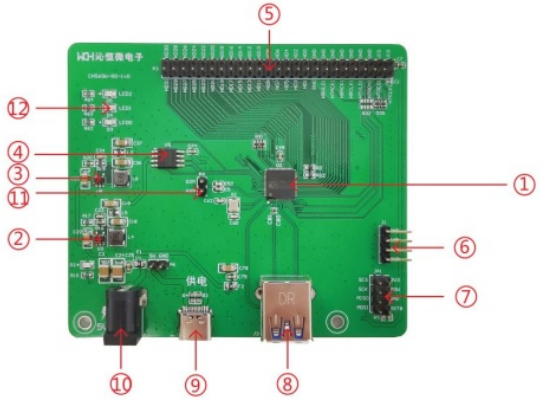
Design a Dev board based on WCH CH569 RISC-V MCU

- At start of this project (January 2022) there was no any “official” available board based on WCH CH569 chipset
- Official WCH Dev Board called "WCH(Jiangsu Qin Heng) CH569W-EVT-R0-1V0 " was not available



- Let’s design a board based on WCH CH569 RISC-V MCU with USB3 & High Speed peripherals

CH569W评估板 \ CH569WEvaluation



模块说明 \ Descriptions

1. 主控MCU	5. HSPI接口	9. Type-C接口
2. 稳压芯片	6. 串口1	10. DC供电插座
3. 稳压芯片	7. ISP下载接口	11. SERDES接口
4. SPI FLASH芯片	8. USB3.0接口	12. LED

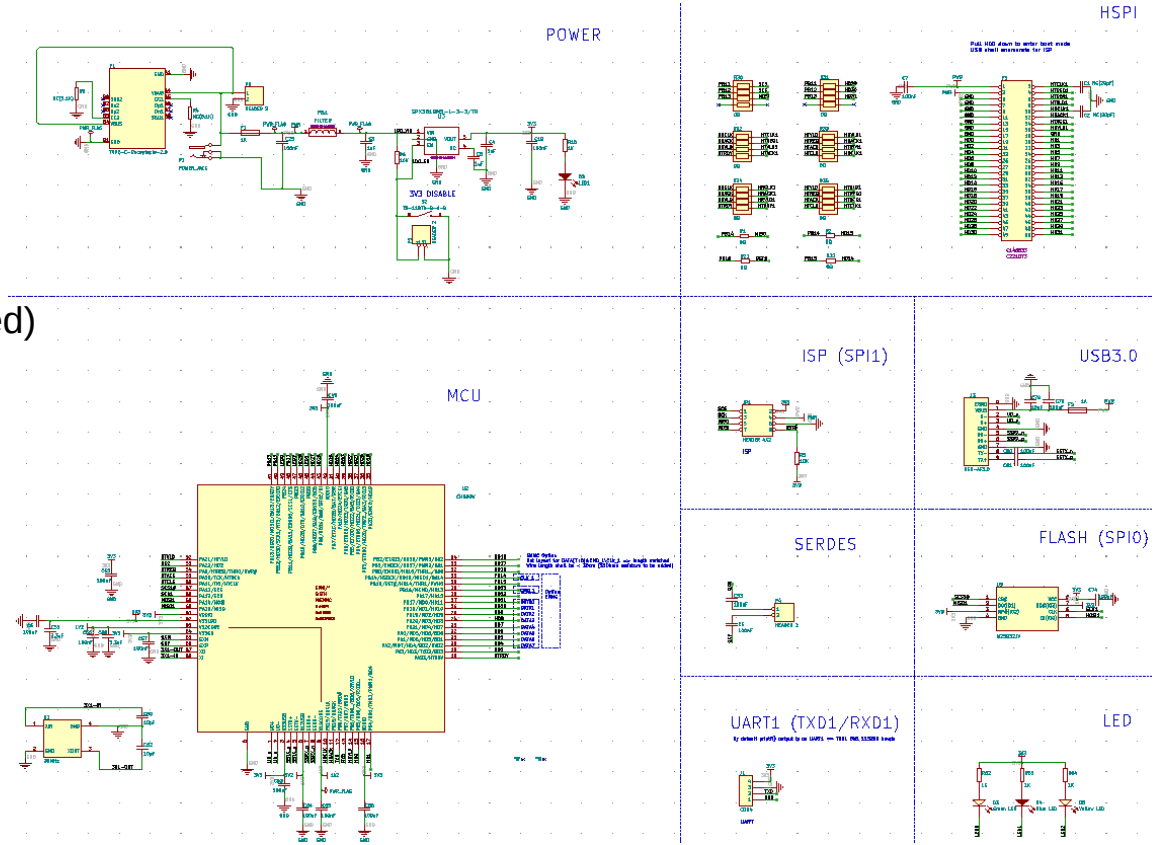
KiCad 6 PCB Design 1st Iteration based on Ref “Altium” Design WCH "CH569W-R0" Schematic

1st Iteration "CH569W-R0"
=> POC for evaluation purpose
4-Layers PCB

<u>Reference Design modifications:</u>
1) Remove 1.2V DC/DC Step-Down(Buck) Converter (not required)

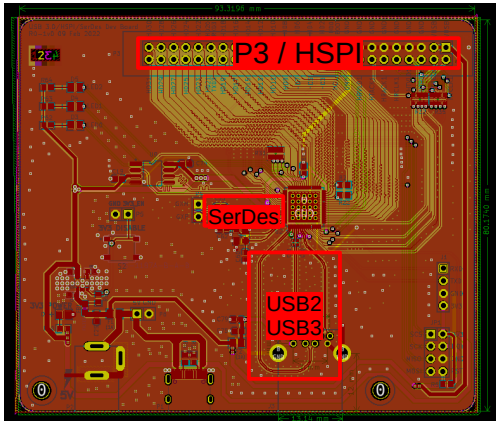
2) Replace 3v3 DC/DC Step-Down(Buck) Converter by LDO for lower noise 40µV / fully protected

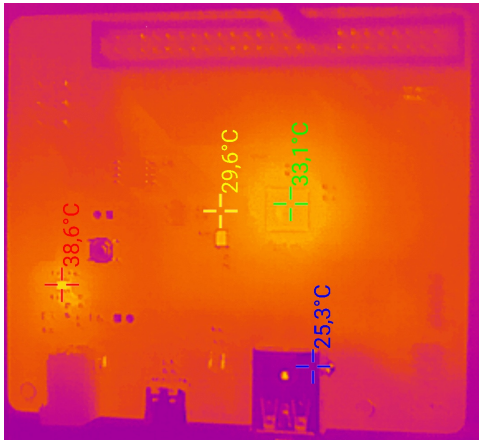
3) Use the best possible Crystal 30MHz (+/-10ppm -40°C/+85°C)

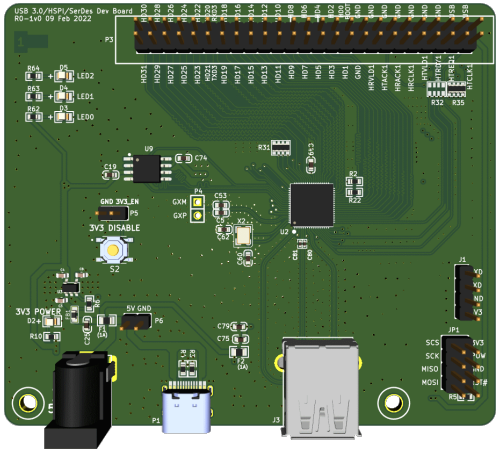


KiCad 6 PCB Design 1st Iteration based on Ref “Altium” Design WCH "CH569W-R0" PCB / Board

- 1st Iteration "CH569W-R0" PCB / Board
- Computed for 4-layers FR4 PCB standard stackup JLCPCB JLC7628
- Match Differential impedance of the USB interface(**USB 2.0&3.0**) **90Ω +/-10%**
- Differential impedance of the **SerDes** interface **100Ω +/-10%**
- Traces length matched **P3 / HSPI** (HDxx signals 56mm)



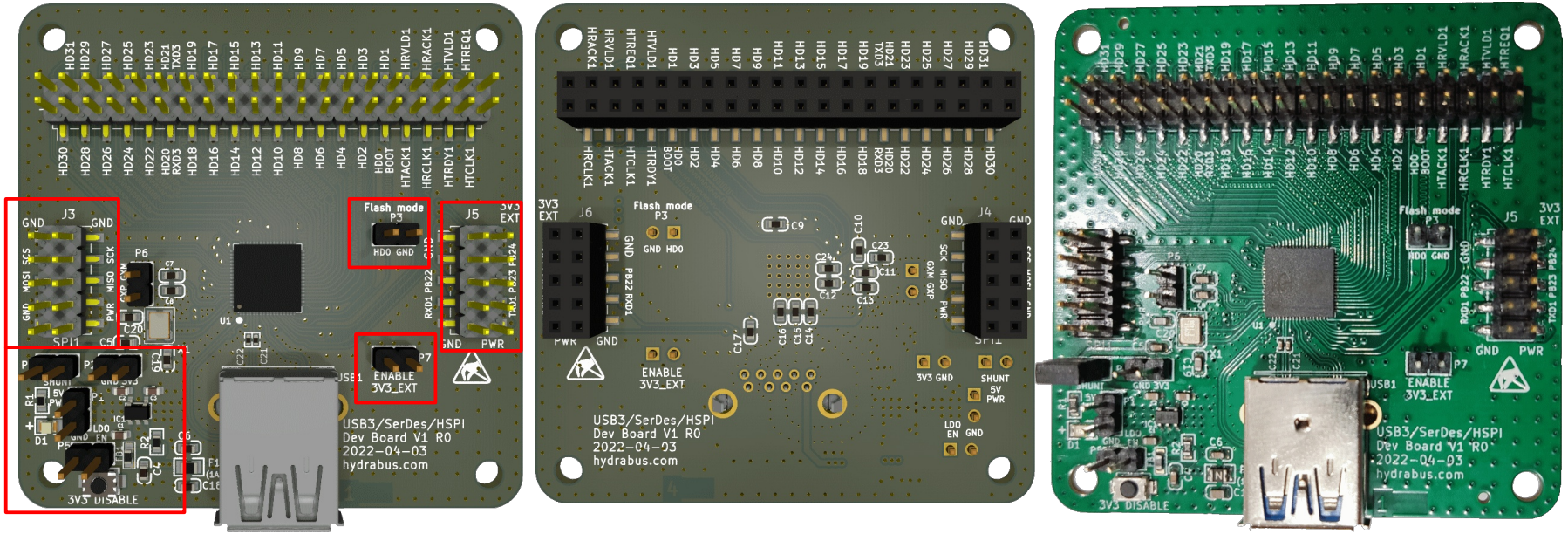




KiCad 6 PCB Design 2nd Iteration “DevBoardCH569W V1 R0“

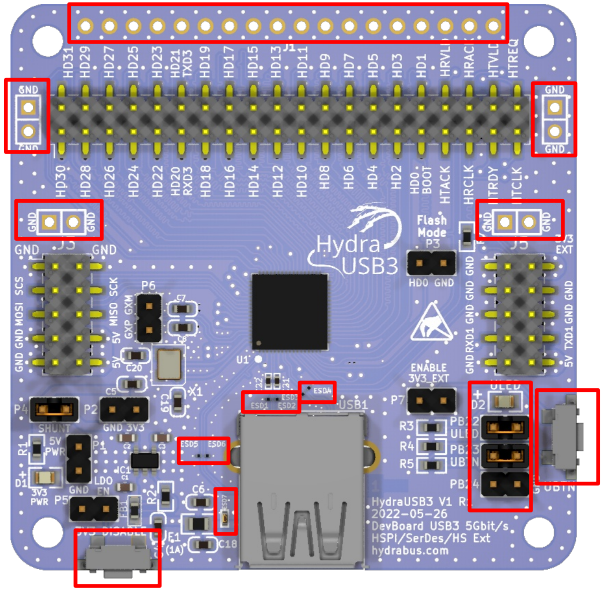
2nd Iteration “DevBoardCH569W V1 R0“

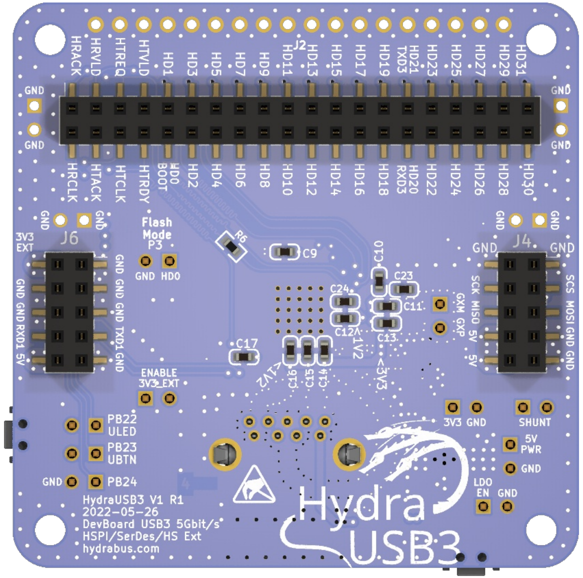
- Full redesign from scratch, more compact 6cm x 6cm, top male pin / bottom female pin
- Designed to plug 2 boards together for DualBoard communication over SPI, HSPI...

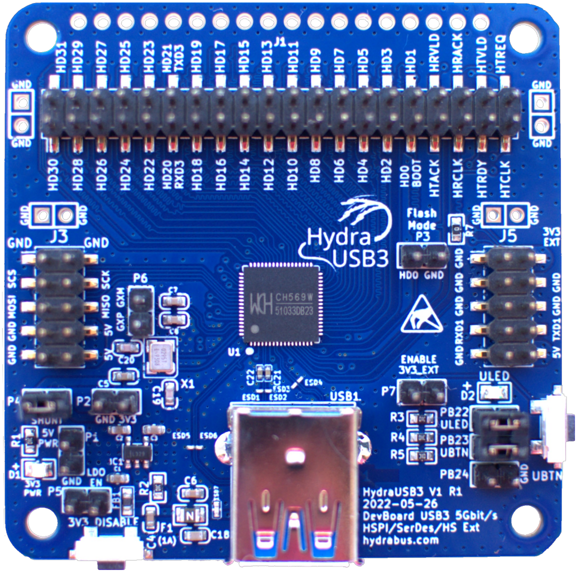


KiCad 6 PCB Design 3rd Iteration “HydraUSB3 V1 R1”

- 3rd Iteration **the birth of “HydraUSB3 V1 R1”**
  - Add RESET(3V3 DISABLE) & UBTN (PB23) switch right angle, ULED, PB24, more GND
  - Add ESD protections on USB2/USB3 & VUSB

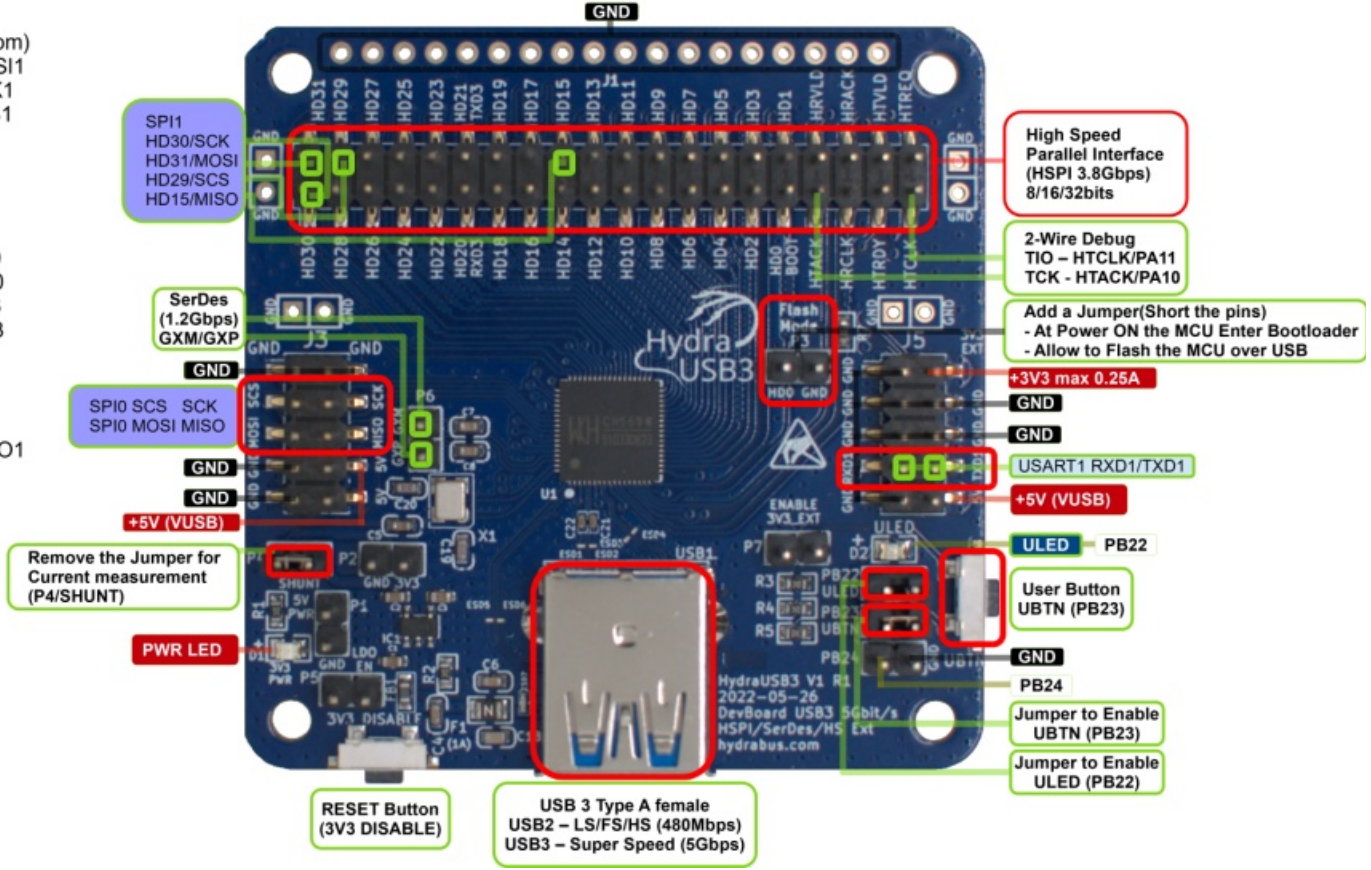






HydraUSB3 V1 R1 final hardware pin assignment

J1 (Top) / J2 (Bottom)
HD31 / PB13 / MOSI1
HD30 / PB12 / SCK1
HD29 / PB11 / SCS1
HD28 / PB10
HD27 / PB9
HD26 / PB8
HD25 / PB7
HD24 / PA16
HD23 / PB6 / TXD0
HD22 / PB5 / RXD0
HD21 / PB4 / TXD3
HD20 / PB3 / RXD3
HD19 / PA20
HD18 / PB2
HD17 / PB1
HD16 / PB0
HD15 / PB14 / MISO1
HD14 / PB15
HD13 / PB16
HD12 / PA17
HD11 / PB17
HD10 / PB18
HD9 / PB19
HD8 / PB20
HD7 / PB21
HD6 / PA0
HD5 / PA1
HD4 / PA2 / RXD2
HD3 / PA3 / TXD2
HD2 / PA22
HD1 / PA4
HD0 / PA5



Reverse Engineering / WCH CH569 Documentation / Datasheet...

- **How it started**

第 16 章 串并互转控制器及收发器（SerDes）

系统内置了支持信号隔离和远距离传输的 SerDes 模块，支持 1.2Gbps 高速差分信号（GXM/GXP 引脚），可以通过光纤模块或网线中的一个差分对等传输媒体，进行远距离数据传输。

SerDes 模块外设基地址：0x4000B000

主要特征

- 可编程的数据
- 内置8b/10b编
- 内置FIFO，支
- 支持DMA功能，
- 提供多种传输
- 实测在600Mbp
- 实测在600Mbp
- 实测在1.2Gbp

具体应用请参考和调用

第 17 章 USB3.0 控制器及收发器（USBSS）

USB3.0 控制器模块包含 USB 主机控制器和 USB 设备控制器功能，搭配系统内置的物理收发器 PHY，可实现 USB3.0 接口产品功能。支持 5Gbps 的 USB SuperSpeed 超高速信号，硬件接口包括 2 对超高速差分信号线（SSRXA/SSRXB 和 SSTXA/SSTXB，A/B 可连+/-或-/+）。

此控制器模块为应用代码提供了链接层寄存器访问接口，用于管理设备的连接和断开、总线状态、电源模式。提供了主机（HOST）功能访问接口，设备（DEVICE）功能访问接口，用于实现 USB3.0 协议规范的各种数据传输及上层协议。

USBSS 模块外设基地址：0x40008000

No details...

主要特征

- 支持USB3.0协议接口规范
- 支持USB Host主机功能和USB Device设备功能
- 支持OTG功能
- 支持驱动USB3.0 HUB
- 主机和设备均支持控制传输、批量传输、中断传输、实时/同步传输
- DMA方式直接访问各端点缓冲区的数据
- 电源管理，支持U1/U2/U3低功耗状态
- 非0端点均支持最大1024字节的数据包，支持突发模式

具体应用请参考和调用提供的子程序库。

- **How it's going** "English Datasheet April 2022"

Chapter 16 Serializer and Deserializer (SerDes)

The system has a built-in SerDes module that supports signal isolation and long-distance transmission. The SerDes module supports 1.2Gbps high-speed differential signals (GXM/GXP pins). Long-distance data transmission can be implemented through a differential peer transmission medium in the optical fiber module or network cable.

SerDes module peripheral base address: 0x4000B000

Main features

- Programmable data reception/transm
- Built-in 8b/10b codec and CRC chec
- Built-in FIFO, which supports dual b
- Support DMA function; access addre
- Provide multiple transmission inte application layer in a timely manne
- Differential transceiving, which can
- The measured transmission distanc network cables at a rate of 600Mbp
- The measured transmission distance cables at a rate of 600Mbps
- The measured transmission distance cables at a rate of 1.2Gbps
- The measured transmission distanc network cables at a rate of 360Mbp

Please refer to and call the provided subro

Chapter 17 USB3.0 controller and transceiver (USBSS)

The USB3.0 controller includes a USB host controller and a USB device controller. With the built-in physical transceiver PHY, it can implement the functions of USB3.0 interface products. It supports 5Gbps USB SuperSpeed signal, and hardware interface includes 2 pairs of super high speed differential signal lines (SSRXA/SSRXB and SSTXA/SSTXB, A/B can be connected to +/- or -/+).

This controller provides a link layer register access interface for application code to manage device connection/disconnection, bus status and power mode. It provides a HOST access interface and a DEVICE access interface, which are used to implement various data transmission and upper layer protocols in the USB3.0 protocol specification.

USBSS module peripheral base address: 0x40008000

Main features

- Supports USB3.0 protocol interface specification
- Supports USB Host function and USB Device function
- Supports OTG function
- Supports drive USB3.0 HUB
- Both the host and the device support control transmission, bulk transmission, interrupt transmission, real-time/synchronous transmission
- Directly access the data of each endpoint buffer through DMA
- Power management, support U1/U2/U3 low power state
- All endpoints other than endpoint 0(zero) support the data packets up to 1024 bytes, and support burst mode

Please refer to and call the provided subroutine library for specific applications.

Reverse Engineering / WCH CH569 Documentation / Datasheet...

- **How it started**

第 16 章 串并互转控制器及收发器（SerDes）

系统内置了支持信号隔离和远距离传输的 SerDes 模块，支持 1.2Gbps 高速差分信号（GXM/GXP 引脚），可以通过光纤模块或网线中的一个差分对等传输媒体，进行远距离数据传输。

SerDes 模块外设基地址：0x4000B000

主要特征

- 可编程的数据
- 内置8b/10b编
- 内置FIFO，支
- 支持DMA功能，
- 提供多种传输
- 实测在600Mbp
- 实测在600Mbp
- 实测在1.2Gbp

具体应用请参考和调用

第 17 章 USB3.0 控制器及收发器（USBSS）

USB3.0 控制器模块包含 USB 主机控制器和 USB 设备控制器功能，搭配系统内置的物理收发器 PHY，可实现 USB3.0 接口产品功能。支持 5Gbps 的 USB SuperSpeed 超高速信号，硬件接口包括 2 对超高速差分信号线（SSRXA/SSRXB 和 SSTXA/SSTXB，A/B 可连+/-或-/+）。

此控制器模块为应用代码提供了链接层寄存器访问接口，用于管理设备的连接和断开、总线状态、电源模式。提供了主机（HOST）功能访问接口，设备（DEVICE）功能访问接口，用于实现 USB3.0 协议规范的各种数据传输及上层协议。

USBSS 模块外设基地址：0x40008000

主要特征

- 支持USB3.0协议接口规范
- 支持USB Host主机功能和USB D
- 支持OTG功能
- 支持驱动USB3.0 HUB
- 主机和设备均支持控制传输、
- DMA方式直接访问各端点缓冲区
- 电源管理，支持U1/U2/U3低功耗
- 非0端点均支持最大1024字节的

具体应用请参考和调用提供的子程序库。

No details...

This is fine

- **How it's going** "English Datasheet April 2022"

Chapter 16 Serializer and Deserializer (SerDes)

The system has a built-in SerDes module that supports signal isolation and long-distance transmission. The SerDes module supports 1.2Gbps high-speed differential signals (GXM/GXP pins). Long-distance data transmission can be implemented through a differential peer transmission medium in the optical fiber module or network cable.

SerDes module peripheral base address: 0x4000B000

Chapter 17 USB3.0 controller and transceiver (USBSS)

Main features

- Programmable data reception/transm
- Built-in 8b/10b codec and CRC chec
- Built-in FIFO, which supports dual b
- Support DMA function; access addre
- Provide multiple transmission inte application layer in a timely manne
- , which can
- sion distanc
- of 600Mbp
- ion distance
- Mbps
- ion distance
- bps
- sion distanc
- of 360Mbp
- ovided subro

The USB3.0 controller includes a USB host controller and a USB device controller. With the built-in physical transceiver PHY, it can implement the functions of USB3.0 interface products. It supports 5Gbps USB SuperSpeed signal, and hardware interface includes 2 pairs of super high speed differential signal lines (SSRXA/SSRXB and SSTXA/SSTXB, A/B can be connected to +/- or -/+).

This controller provides a link layer register access interface for application code to manage device connection/disconnection, bus status and power mode. It provides a HOST access interface and a DEVICE access interface, which are used to implement various data transmission and upper layer protocols in the USB3.0 protocol specification.

USBSS module peripheral base address: 0x40008000

Main features

- Supports USB3.0 protocol interface specification
- Supports USB Host function and USB Device function
- Supports OTG function
- Supports drive USB3.0 HUB
- Both the host and the device support control transmission, bulk transmission, interrupt transmission, real-time/synchronous transmission
- Directly access the data of each endpoint buffer through DMA
- Power management, support U1/U2/U3 low power state
- All endpoints other than endpoint 0(zero) support the data packets up to 1024 bytes, and support burst mode

Please refer to and call the provided subroutine library for specific applications.

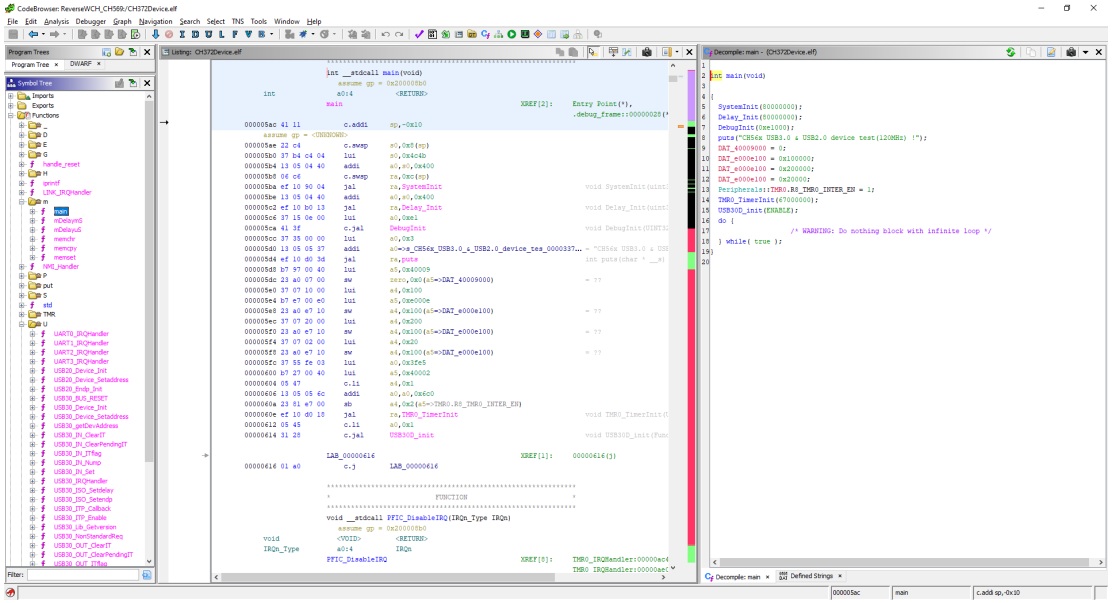
Firmware Reverse Engineering RISC-V tools

Does there is any good reverse engineering tool for RISC-V with decompiler feature ?

- JEB Decompiler for RISC-V is not free (not tested / evaluation version is ultra limited)
- IDA Pro is not free and does not support decompiler for RISC-V

Free alternative supporting RISC-V

- Radare 2
- Ghidra 10 a must have !!
  - **Support RISC-V decompiler**



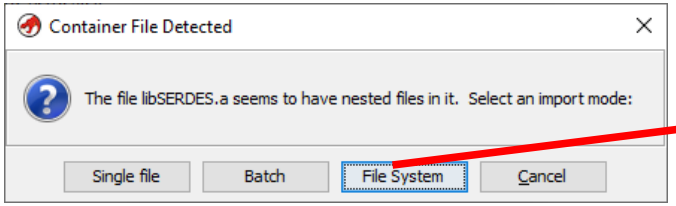
Firmware Reverse Engineering RISC-V SerDes Blob intro

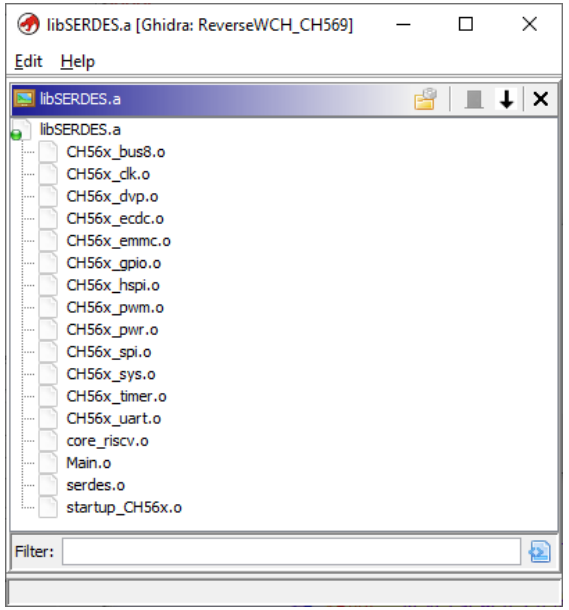
WCH CH569W SerDes PHY (Serializer/Deserializer) at 1.20Gbps officially with a “**static lib/blob**”

- Can be found online
  https://github.com/SoCXin/CH569/tree/master/src/EVT/EXAM/SERDES/User
- Removed from official WCH CH569 repository
  https://github.com/openwch/ch569/tree/main/EVT/EXAM
- **Why to reverse it ?**
  - Have fully open source code **“without blob”**
  - Support different SerDes speed and improve the driver

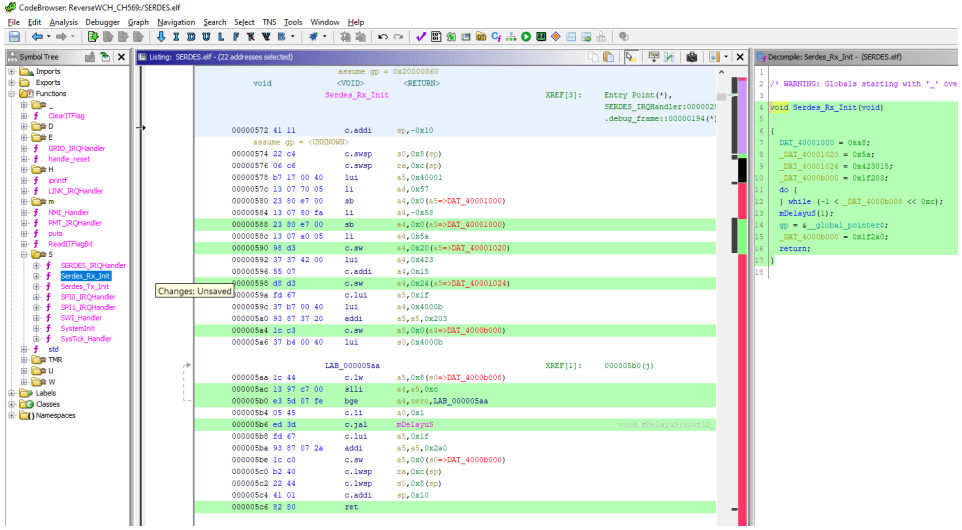
Firmware Reverse Engineering RISC-V SerDes Blob with Ghidra

How to reverse a library blob ?



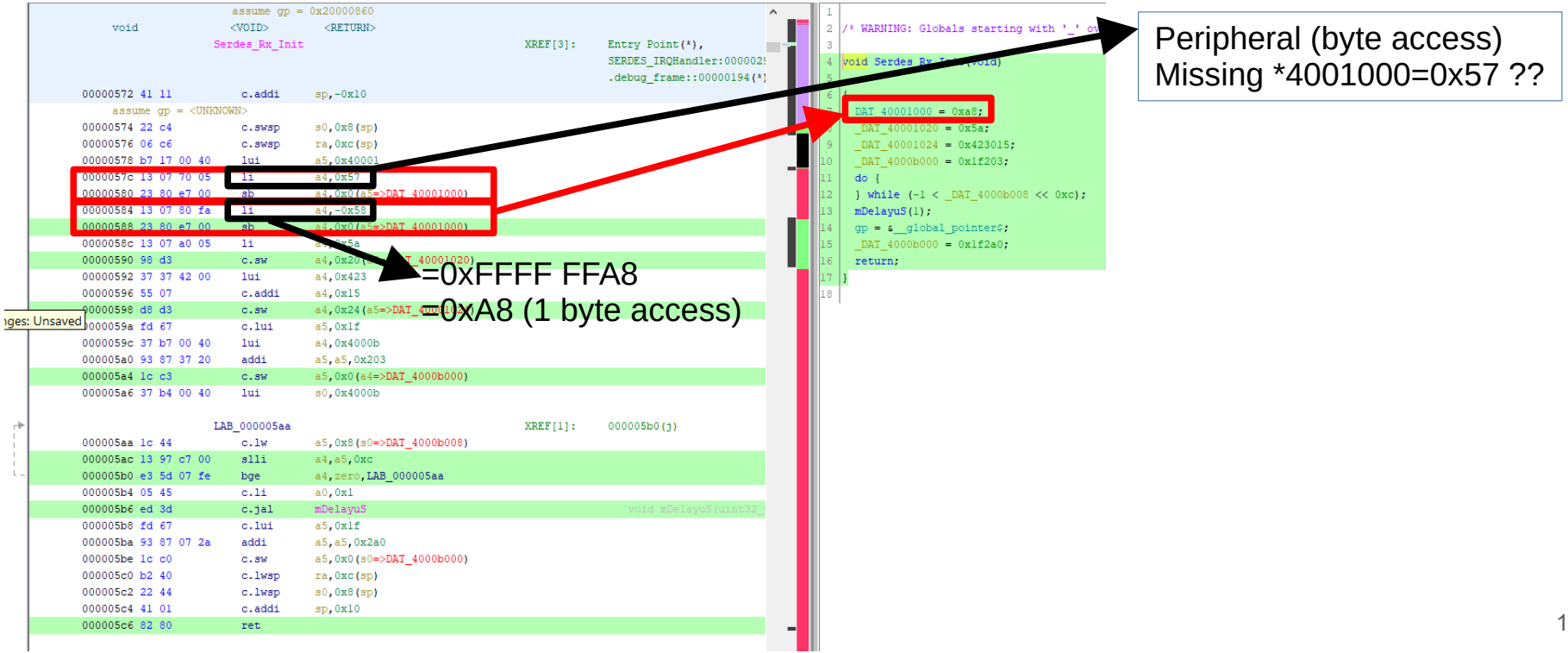


The best/simple way:
Build an executable(with debug) using the library



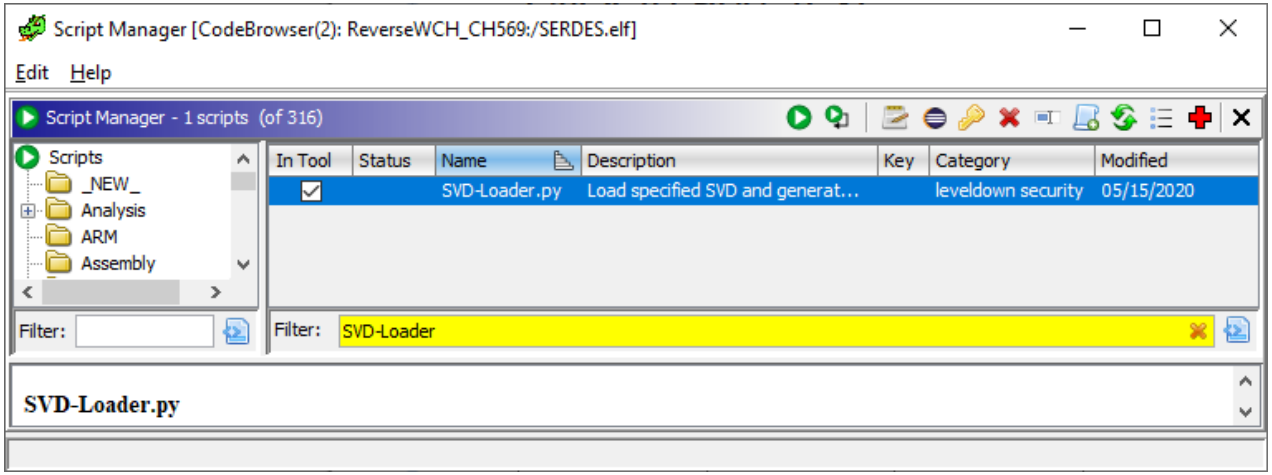
Firmware Reverse Engineering RISC-V SerDes Blob Ghidra Peripheral access Pseudo C code bug

SerDes RX Init reverse engineering with Ghidra (RISC-V Peripheral access Pseudo C code bug)



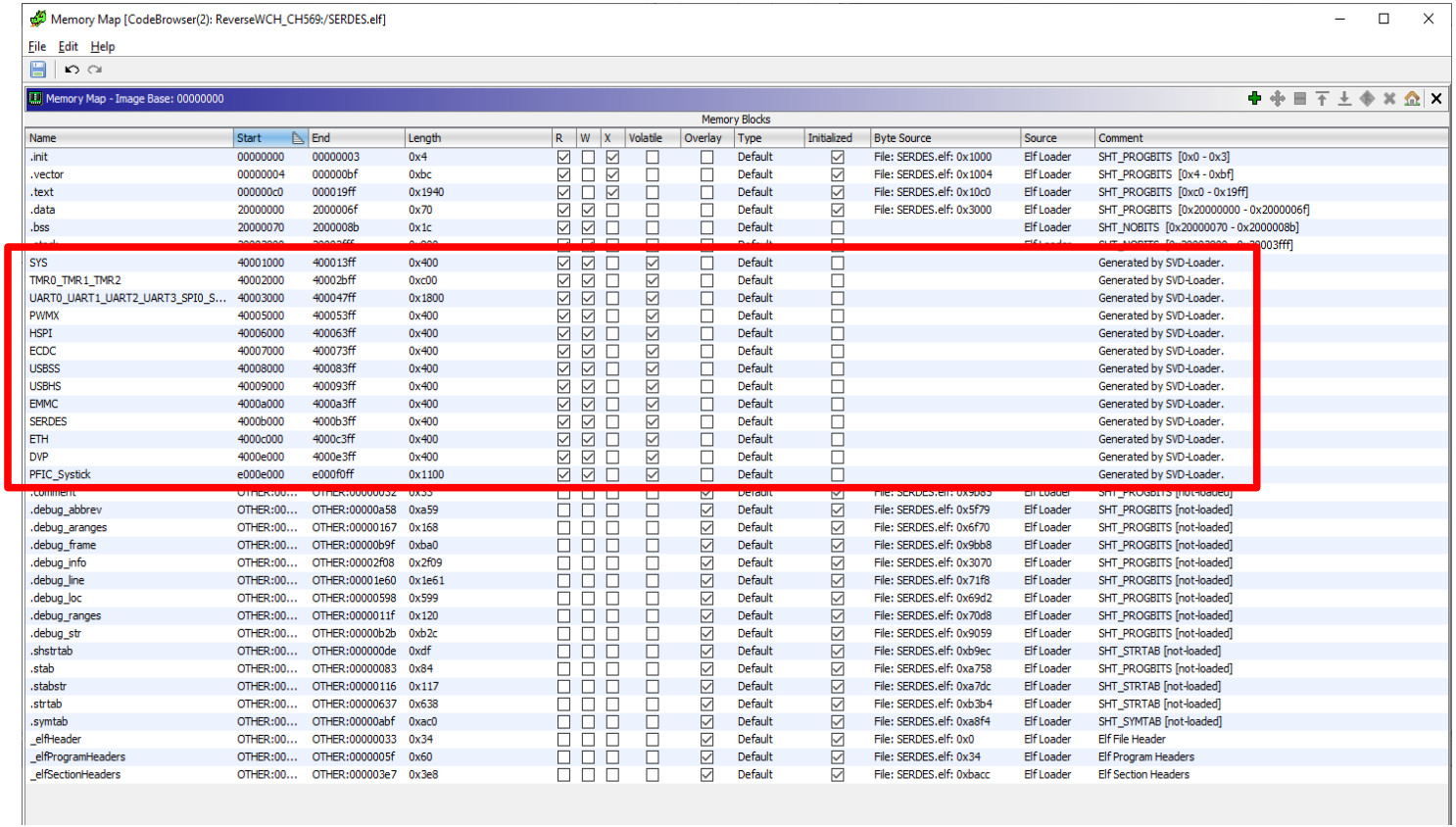
Firmware Reverse Engineering RISC-V SerDes Blob Ghidra ASM / Decompiler bug Workaround

Use **SVD-Loader.py** with **CH56Xxx.svd** (provided in MounRiver Studio Community v1.30)
(SVD = System View Description)



https://github.com/leveldown-security/SVD-Loader-Ghidra

Firmware Reverse Engineering RISC-V SerDes Blob Ghidra CH56x SVD

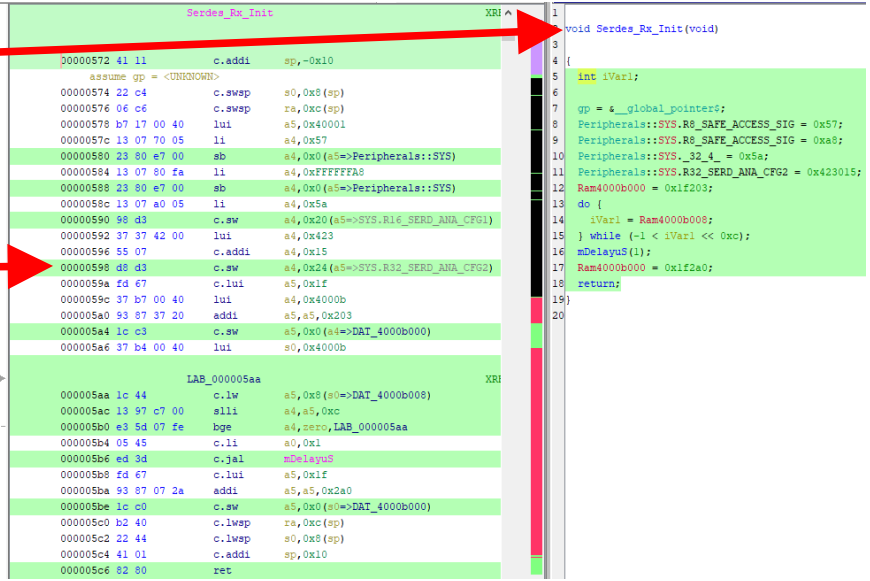


Name	Start	End	Length	R	W	X	Volatile	Overlay	Type	Initialized	Byte Source	Source	Comment
.init	00000000	00000003	0x4	☑	☐	☑	☐	☐	Default	☑	File: SERDES.elf: 0x1000	Elf Loader	SHT_PROGBITS [0x0 - 0x3]
.vector	00000004	000000bf	0xbc	☑	☐	☑	☐	☐	Default	☑	File: SERDES.elf: 0x1004	Elf Loader	SHT_PROGBITS [0x4 - 0xbf]
.text	000000c0	000019ff	0x1940	☑	☐	☑	☐	☐	Default	☑	File: SERDES.elf: 0x10c0	Elf Loader	SHT_PROGBITS [0xc0 - 0x19ff]
.data	20000000	2000006f	0x70	☑	☑	☐	☐	☐	Default	☑	File: SERDES.elf: 0x3000	Elf Loader	SHT_PROGBITS [0x20000000 - 0x2000006f]
.bss	20000070	2000008b	0x1c	☑	☑	☐	☐	☐	Default	☐		Elf Loader	SHT_NOBITS [0x20000070 - 0x2000008b]
SYS	40001000	400013ff	0x400	☑	☑	☐	☑	☐	Default	☐			Generated by SVD-Loader.
TMR0_TMR1_TMR2	40002000	40002bff	0xc00	☑	☑	☐	☑	☐	Default	☐			Generated by SVD-Loader.
UART0_UART1_UART2_UART3_SPI0_S...	40003000	400047ff	0x1800	☑	☑	☐	☑	☐	Default	☐			Generated by SVD-Loader.
PWMX	40005000	400053ff	0x400	☑	☑	☐	☑	☐	Default	☐			Generated by SVD-Loader.
HSPI	40006000	400063ff	0x400	☑	☑	☐	☑	☐	Default	☐			Generated by SVD-Loader.
ECDC	40007000	400073ff	0x400	☑	☑	☐	☑	☐	Default	☐			Generated by SVD-Loader.
USBSS	40008000	400083ff	0x400	☑	☑	☐	☑	☐	Default	☐			Generated by SVD-Loader.
USBHS	40009000	400093ff	0x400	☑	☑	☐	☑	☐	Default	☐			Generated by SVD-Loader.
EMMC	4000a000	4000a3ff	0x400	☑	☑	☐	☑	☐	Default	☐			Generated by SVD-Loader.
SERDES	4000b000	4000b3ff	0x400	☑	☑	☐	☑	☐	Default	☐			Generated by SVD-Loader.
ETH	4000c000	4000c3ff	0x400	☑	☑	☐	☑	☐	Default	☐			Generated by SVD-Loader.
DVP	4000e000	4000e3ff	0x400	☑	☑	☐	☑	☐	Default	☐			Generated by SVD-Loader.
PFIC_Systick	e000e000	e000f0ff	0x1100	☑	☑	☐	☑	☐	Default	☐			Generated by SVD-Loader.
.debug_abbrev	OTHER:00...	OTHER:00000a58	0xa59	☐	☐	☐	☐	☑	Default	☑	File: SERDES.elf: 0x5f79	Elf Loader	SHT_PROGBITS [not-loaded]
.debug_aranges	OTHER:00...	OTHER:00000167	0x168	☐	☐	☐	☐	☑	Default	☑	File: SERDES.elf: 0x6f70	Elf Loader	SHT_PROGBITS [not-loaded]
.debug_frame	OTHER:00...	OTHER:00000b9f	0xba0	☐	☐	☐	☐	☑	Default	☑	File: SERDES.elf: 0x9bb8	Elf Loader	SHT_PROGBITS [not-loaded]
.debug_info	OTHER:00...	OTHER:00002f08	0x2f09	☐	☐	☐	☐	☑	Default	☑	File: SERDES.elf: 0x3070	Elf Loader	SHT_PROGBITS [not-loaded]
.debug_line	OTHER:00...	OTHER:00001e60	0x1e61	☐	☐	☐	☐	☑	Default	☑	File: SERDES.elf: 0x71f8	Elf Loader	SHT_PROGBITS [not-loaded]
.debug_loc	OTHER:00...	OTHER:00000598	0x599	☐	☐	☐	☐	☑	Default	☑	File: SERDES.elf: 0x69d2	Elf Loader	SHT_PROGBITS [not-loaded]
.debug_ranges	OTHER:00...	OTHER:0000011f	0x120	☐	☐	☐	☐	☑	Default	☑	File: SERDES.elf: 0x70d8	Elf Loader	SHT_PROGBITS [not-loaded]
.debug_str	OTHER:00...	OTHER:00000b2b	0xb2c	☐	☐	☐	☐	☑	Default	☑	File: SERDES.elf: 0x9059	Elf Loader	SHT_PROGBITS [not-loaded]
.shstrtab	OTHER:00...	OTHER:000000de	0xdf	☐	☐	☐	☐	☑	Default	☑	File: SERDES.elf: 0xb9ec	Elf Loader	SHT_STRTAB [not-loaded]
.stab	OTHER:00...	OTHER:00000083	0x84	☐	☐	☐	☐	☑	Default	☑	File: SERDES.elf: 0xa758	Elf Loader	SHT_PROGBITS [not-loaded]
.stabstr	OTHER:00...	OTHER:00000116	0x117	☐	☐	☐	☐	☑	Default	☑	File: SERDES.elf: 0xa7dc	Elf Loader	SHT_STRTAB [not-loaded]
.strtab	OTHER:00...	OTHER:00000637	0x638	☐	☐	☐	☐	☑	Default	☑	File: SERDES.elf: 0xb3b4	Elf Loader	SHT_STRTAB [not-loaded]
.symtab	OTHER:00...	OTHER:00000abf	0xac0	☐	☐	☐	☐	☑	Default	☑	File: SERDES.elf: 0xa8f4	Elf Loader	SHT_SYMTAB [not-loaded]
_elfHeader	OTHER:00...	OTHER:00000033	0x34	☐	☐	☐	☐	☑	Default	☑	File: SERDES.elf: 0x0	Elf Loader	Elf File Header
_elfProgramHeaders	OTHER:00...	OTHER:0000005f	0x60	☐	☐	☐	☐	☑	Default	☑	File: SERDES.elf: 0x34	Elf Loader	Elf Program Headers
_elfSectionHeaders	OTHER:00...	OTHER:000003e7	0x3e8	☐	☐	☐	☐	☑	Default	☑	File: SERDES.elf: 0xbacc	Elf Loader	Elf Section Headers

Firmware Reverse Engineering RISC-V SerDes Blob Ghidra ASM / Pseudo C code bug Workaround

Before applying SVD-Loader with **CH56Xxx.svd**

After applying SVD-Loader with **CH56Xxx.svd**



Firmware Reverse Engineering SerDes Blob Ghidra Serdes_Rx_Init

Ghidra 10.2 Disassembly/Pseudo C original SerDes_Rx blob **Serdes_Rx_Init**

```
                        Serdes_Rx_Init                                  XRI
00000572 41 11           c.addi     sp,-0x10
     assume gp = <UNKNOWN>
00000574 22 c4           c.swsp     s0,0x8(sp)
00000576 06 c6           c.swsp     ra,0xc(sp)
00000578 b7 17 00 40     lui        a5,0x40001
0000057c 13 07 70 05     li         a4,0x57
00000580 23 80 e7 00     sb         a4,0x0(a5=>Peripherals::SYS)
00000584 13 07 80 fa     li         a4,0xFFFFFFA8
00000588 23 80 e7 00     sb         a4,0x0(a5=>Peripherals::SYS)
0000058c 13 07 a0 05     li         a4,0x5a
00000590 98 d3           c.sw       a4,0x20(a5=>SYS.R16_SERD_ANA_CFG1)
00000592 37 37 42 00     lui        a4,0x423
00000596 55 07           c.addi     a4,0x15
00000598 d8 d3           c.sw       a4,0x24(a5=>SYS.R32_SERD_ANA_CFG2)
0000059a fd 67           c.lui      a5,0x1f
0000059c 37 b7 00 40     lui        a4,0x4000b
000005a0 93 87 37 20     addi       a5,a5,0x203
000005a4 1c c3           c.sw       a5,0x0(a4=>DAT_4000b000)
000005a6 37 b4 00 40     lui        s0,0x4000b

                        LAB_000005aa                                    XRI
000005aa 1c 44           c.lw       a5,0x8(s0=>DAT_4000b008)
000005ac 13 97 c7 00     slli       a4,a5,0xc
000005b0 e3 5d 07 fe     bge        a4,zero,LAB_000005aa
000005b4 05 45           c.li       a0,0x1
000005b6 ed 3d           c.jal      mDelayuS
000005b8 fd 67           c.lui      a5,0x1f
000005ba 93 87 07 2a     addi       a5,a5,0x2a0
000005be 1c c0           c.sw       a5,0x0(s0=>DAT_4000b000)
000005c0 b2 40           c.lwsp     ra,0xc(sp)
000005c2 22 44           c.lwsp     s0,0x8(sp)
000005c4 41 01           c.addi     sp,0x10
000005c6 82 80           ret
```

```
1
2  void Serdes_Rx_Init(void)
3
4  {
5    int iVar1;
6
7    gp = &__global_pointer$;
8    Peripherals::SYS.R8_SAFE_ACCESS_SIG = 0x57;
9    Peripherals::SYS.R8_SAFE_ACCESS_SIG = 0xa8;
10   Peripherals::SYS._32_4_ = 0x5a;
11   Peripherals::SYS.R32_SERD_ANA_CFG2 = 0x423015;
12   Ram4000b000 = 0x1f203;
13   do {
14     iVar1 = Ram4000b008;
15   } while (-1 < iVar1 << 0xc);
16   mDelayuS(1);
17   Ram4000b000 = 0x1f2a0;
18   return;
19 }
20
```

Final C code with improvements

```
void SerDes_Rx_Init(e_sds_pll_freq SDS_PLL_FREQ)
{
    uint32_t sds_status;

    R8_SAFE_ACCESS_SIG = 0x57; // enable safe access mode
    R8_SAFE_ACCESS_SIG = 0xa8;
    R16_SERD_ANA_CFG1 = 0x5a;
    R32_SERD_ANA_CFG2 = 0x00423015;

    SDS->SDS_CTRL = (PHY_RESET | LINK_RESET | SDS_ALL_CLR);
    bsp_wait_us_delay(1);

    /* R32_SERD_ANA_CFG2
     * [24:0] RB_SERD_TRX_CFG
     * Receive/transmit parameter configuration of SerDes PHY.
     */
    SDS->SDS_CTRL = (SDS_POWR_UP |
                     SDS_TX_PU | SDS_RX_PU |
                     SDS_PLL_PU | SDS_PLL_FREQ |
                     LINK_RESET | SDS_ALL_CLR);

    /* Wait until SDS PLL is ready */
    do
    {
        sds_status = SDS->SDS_STATUS;
    }
    while(!(sds_status & SDS_PLL_READY));

    bsp_wait_us_delay(1);

    SDS->SDS_CTRL = (SDS_POWR_UP |
                     SDS_TX_PU | SDS_RX_PU |
                     SDS_PLL_PU | SDS_PLL_FREQ |
                     SDS_DMA_EN | SDS_RX_EN);
}
```

Firmware Reverse Engineering SerDes Blob Ghidra Serdes_Tx_Init

Ghidra 10.2 Disassembly/Pseudo C original SerDes_Tx blob **Serdes_Tx_Init**

```
                    void __stdcall Serdes_Tx_Init(void)
                       assume gp = 0x20000860
     void              <VOID>          <RETURN>
                    Serdes_Tx_Init                                  XR

0000052c b7 17 00 40     lui         a5,0x40001
     assume gp = <UNKNOWN>
00000530 13 07 70 05     li          a4,0x57
00000534 23 80 e7 00     sb          a4,0x0(a5=>Peripherals::SYS)
00000538 13 07 80 fa     li          a4,-0x58
0000053c 23 80 e7 00     sb          a4,0x0(a5=>Peripherals::SYS)
00000540 13 07 a0 05     li          a4,0x5a
00000544 98 d3           c.sw        a4,0x20(a5=>SYS.R16_SERD_ANA_CFG1)
00000546 37 37 42 00     lui         a4,0x423
0000054a 55 07           c.addi      a4,0x15
0000054c d8 d3           c.sw        a4,0x24(a5=>SYS.R32_SERD_ANA_CFG2)
0000054e fd 67           c.lui       a5,0x1f
00000550 37 b7 00 40     lui         a4,0x4000b
00000554 93 87 37 20     addi        a5,a5,0x203
00000558 1c c3           c.sw        a5,0x0(a4=>DAT_4000b000)

                    LAB_0000055a                                    XR
0000055a 1c 47           c.lw        a5,0x8(a4=>DAT_4000b008)
0000055c 93 96 c7 00     slli        a3,a5,0xc
00000560 e3 dd 06 fe     bge         a3,zero,LAB_0000055a
00000564 b7 f7 07 00     lui         a5,0x7f
00000568 93 87 87 2c     addi        a5,a5,0x2c8
0000056c 1c c3           c.sw        a5,0x0(a4=>DAT_4000b000)
0000056e 05 45           c.li        a0,0x1
00000570 81 b7           c.j         mDelayuS
                    -- Flow Override: CALL_RETURN (CALL_TERMINATOR
```

```
1
2  void Serdes_Tx_Init(void)
3
4  {
5    int iVar1;
6
7    gp = &__global_pointer$;
8    Peripherals::SYS.R8_SAFE_ACCESS_SIG = 0x57;
9    Peripherals::SYS.R8_SAFE_ACCESS_SIG = 0xa8;
10   Peripherals::SYS._32_4_ = 0x5a;
11   Peripherals::SYS.R32_SERD_ANA_CFG2 = 0x423015;
12   DAT_4000b000 = 0x1f203;
13   do {
14     iVar1 = DAT_4000b008;
15   } while (-1 < iVar1 << 0xc);
16   DAT_4000b000 = 0x7f2c8;
17   mDelayuS(1);
18   return;
19 }
20
```

Final C code with improvements

```
void SerDes_Tx_Init(e_sds_pll_freq SDS_PLL_FREQ)
{
    uint32_t sds_status;

    R8_SAFE_ACCESS_SIG = 0x57; // enable safe access mode
    R8_SAFE_ACCESS_SIG = 0xa8;
    R16_SERD_ANA_CFG1 = 0x5a;
    R32_SERD_ANA_CFG2 = 0x00423015; // This value is not documented
    /* R32_SERD_ANA_CFG2
     * [24:0] RB_SERD_TRX_CFG (Undocumented)
     * Receive/transmit parameter configuration of SerDes PHY.
     */

    SDS->SDS_CTRL = (PHY_RESET | LINK_RESET | SDS_ALL_CLR);
    SDS->SDS_CTRL = (SDS_POWR_UP |
                     SDS_TX_PU | SDS_RX_PU |
                     SDS_PLL_PU | SDS_PLL_FREQ |
                     LINK_RESET | SDS_ALL_CLR);
    do
    {
        sds_status = SDS->SDS_STATUS;
    }
    while ( !(sds_status & SDS_PLL_READY) );

    bsp_wait_us_delay(1);

    SDS->SDS_CTRL = (SDS_POWR_UP |
                     SDS_TX_PU | SDS_RX_PU |
                     SDS_PLL_PU | SDS_PLL_FREQ |
                     SDS_INT_BUSY_EN |
                     SDS_DMA_EN |
                     SDS_TX_EN |
                     SDS_ALIGN_EN |
                     SDS_CONT_EN);

    SDS->SDS_RTX_CTRL = SDS_LINK_INIT;
    bsp_wait_us_delay(1);

    SDS->SDS_RTX_CTRL = 0;
    do
    {
        sds_status = SDS->SDS_STATUS;
    }
    while(!(sds_status & SDS_TX_READY));
}
```

Code improvement wait TX is ready

Firmware Reverse Engineering SerDes Blob Ghidra Wait_Txdone

Ghidra 10.2 Disassembly/Pseudo C original SerDes blob
Wait_Txdone

```
                    void __stdcall Wait_Txdone(void)
                       assume gp = 0x20000860
     void                <VOID>          <RETURN>
                    Wait_Txdone

0000064a b7 b7 00 40      lui         a5,0x4000b

     assume gp = <UNKNOWN>
                    LAB_0000064e
0000064e 98 47            c.lw        a4,0x8(a5=>DAT_4000b008)
00000650 09 8b            c.andi      a4,0x2
00000652 75 df            c.beqz      a4,LAB_0000064e
00000654 d8 47            c.lw        a4,0xc(a5=>DAT_4000b00c)
00000656 81 76            c.lui       a3,0x20
00000658 fd 16            c.addi      a3,-0x1
0000065a 75 8f            c.and       a4,a3
0000065c d8 c7            c.sw        a4,0xc(a5=>DAT_4000b00c)
0000065e 37 b7 00 40      lui         a4,0x4000b

                    LAB_00000662
00000662 5c 47            c.lw        a5,0xc(a4=>DAT_4000b00c)
00000664 93 96 e7 00      slli        a3,a5,0xe
00000668 e3 cd 06 fe      blt         a3,zero,LAB_00000662
0000066c 82 80            ret
```

```
1
2  void Wait_Txdone(void)
3
4  {
5    uint uVar1;
6    int iVar2;
7
8    gp = &__global_pointer$;
9    do {
10     uVar1 = DAT_4000b008;
11   } while ((uVar1 & 2) == 0);
12   uVar1 = DAT_4000b00c;
13   DAT_4000b00c = uVar1 & 0xfffdffff;
14   do {
15     iVar2 = DAT_4000b00c;
16   } while (iVar2 << 0xe < 0);
17   return;
18 }
19
```

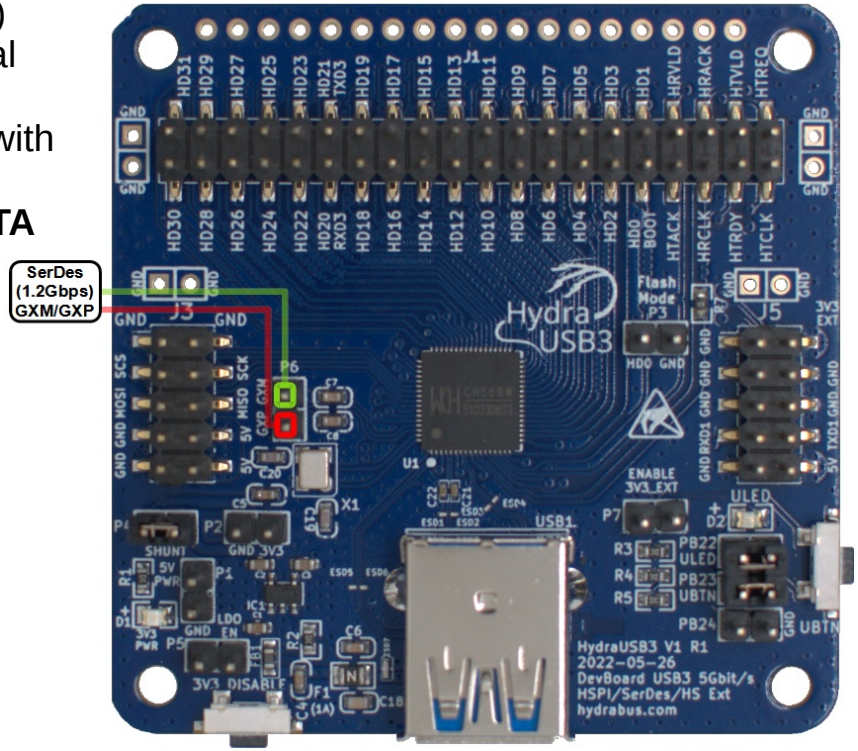
Final C code with improvements

```
void SerDes_Wait_Txdone(void)
{
    uint32_t sds_status;
    uint32_t sds_rtx_ctrl;

    do
    {
        sds_status = SDS->SDS_STATUS;
    }
    while((sds_status & SDS_TX_INT_FLG) == 0);

    SDS->SDS_RTX_CTRL &= ~SDS_TX_START;
    do
    {
        sds_rtx_ctrl = SDS->SDS_RTX_CTRL;
    }
    while(sds_rtx_ctrl & SDS_TX_START);

    /* Clear TX Interrupt Flag */
    SDS->SDS_STATUS = SDS_TX_INT_FLG;
}
```

Code improvement on
TX IRQ

WCH CH569 SerDes Hardware

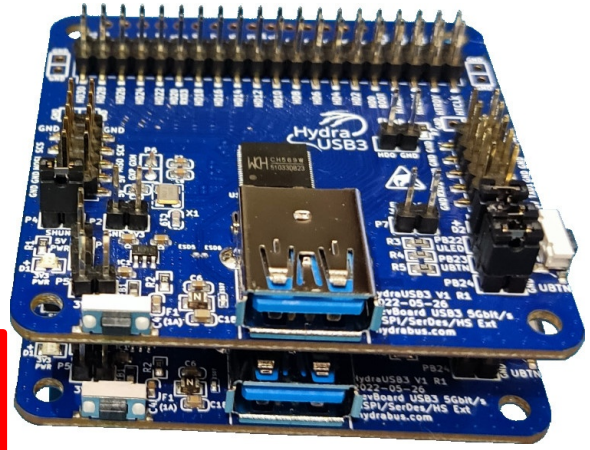
- Default speed **1.2Gbps** (programmable TX/RX rate)
- Based on **LVDS**, only 2 wires (differential pair, signal about +/- 400mV)
  - Can **exceed 90m(<600Mbps)** with **CAT6 cable** with differential line pairs
- **Built-in PHY** with **8b/10b codec** + **CRC check(SATA CRC32)**
- For more details see https://github.com/hydrausb3/wch-ch569-serdes



Firmware Reverse Engineering SerDes Firmware/Host example

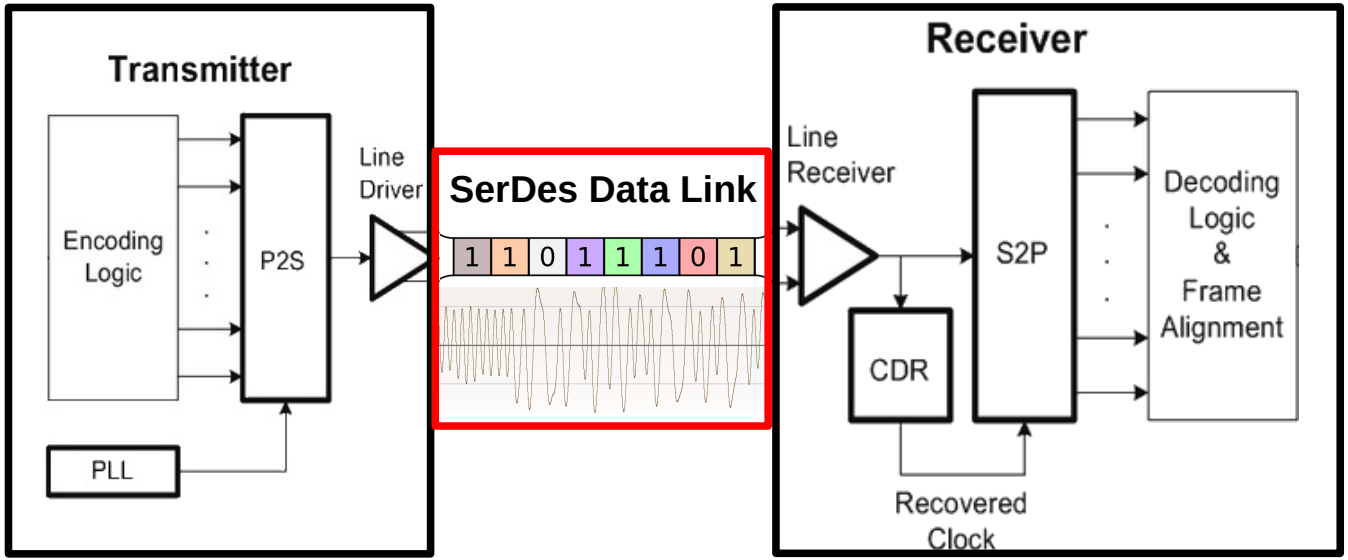
- Reverse & rewrite in C the SerDes driver => DONE
  - SerDes Driver
    https://github.com/hydrausb3/wch-ch56x-bsp/blob/main/drv/CH56x_serdes.c
  - Example SerDes TX/RX (with Dual HydraUSB3+FTDI C232HM-DDHSL-0 @5Mbauds)
    https://github.com/hydrausb3/hydrausb3_fw/tree/main/HydraUSB3_DualBoard_SerDes

```
00s 000ms 020us SYNC 00000001
00s 000ms 000us Start
00s 000ms 076us SerDes_Rx 2022/08/20 @ChipID=69
00s 000ms 221us FSYS=120000000
00s 000ms 327us SerDes_DoubleDMA_Rx_CFG() Before
00s 000ms 468us RX_DMA0_addr=0x20022000 RX_DMA1_addr=0x20021000
00s 000ms 664us SerDes_DoubleDMA_Rx_CFG() After
00s 000ms 800us SerDes_Rx_Init(SERDES_TX_RX_SPEED=0x1200) Before
00s 001ms 262us SerDes_Rx_Init() After
00s 001ms 378us SerDes_EnableIT(SDS_RX_INT_EN|SDS_RX_ERR_EN|SDS_FIFO_OV_EN) Before
00s 001ms 598us SerDes_EnableIT(SDS_RX_INT_EN|SDS_RX_ERR_EN|SDS_FIFO_OV_EN) After
00s 004ms 822us SDS_RX_LEN0=4 SDS_RX_LEN1=4 CNT_nb_cycles=135(1us)
00s 005ms 038us SDS_STATUS[0]=0x010F0024 SDS_STATUS[1]=0x020F0004 SDS_DATA0=0x0FFFFFFF SDS_DATA1=0x1FFFFFFF
00s 005ms 360us SDS_RX_ERR=0 SDS_FIFO_OV=0 RX_CRC_OK=1
00s 005ms 536us 5A5A5A5A
00s 005ms 628us 5A5A5A5A
...
```



Reverse Engineering WCH CH569 SerDes Data Link

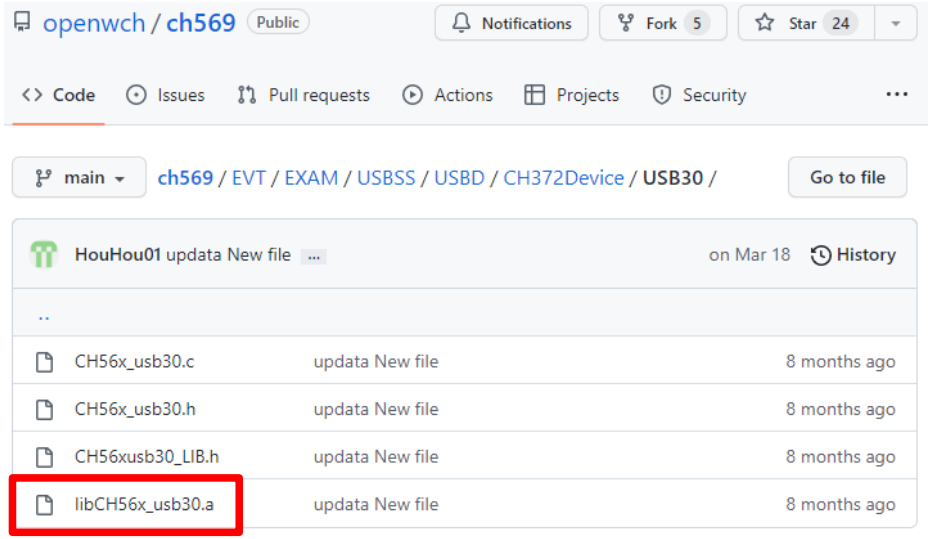
Let's Reverse the **SerDes Data Link**
Why ? => To interface **WCH CH569 SerDes with an FPGA / External world**



CH569W SerDes is based on **Serial ATA (SATA) specification**
Thanks to Andrew Zonenberg @azonenberg for the hint

Firmware Reverse Engineering USB3

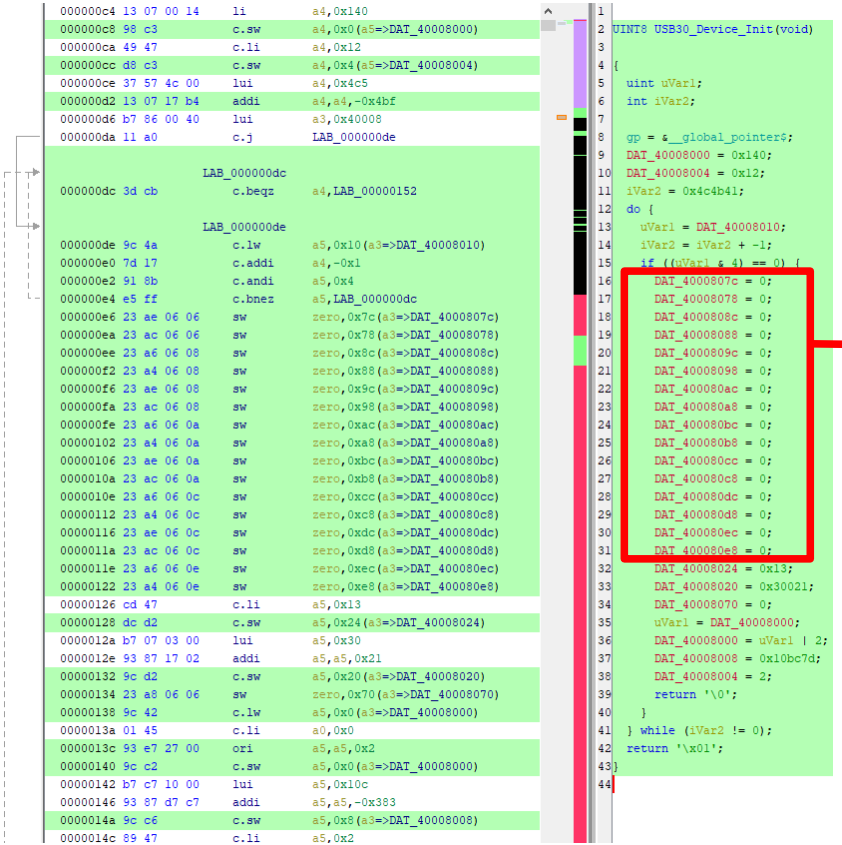
Why? => To support USB3 PHY at 5Gbps officially without using a “**static lib (blob)**”
=> Can be found online on official WCH CH569 repository
https://github.com/openwch/ch569/tree/main/EVT/EXAM/USBSS
=>Contains **USB3 Device (USBD)** and **USB3 Host(USBH)** examples with **blob**
We will focus on the reverse engineering of the **USB3 USBD** Device “blob **libCH56x_usb30.a**” based on “**CH372Device Example**” to use the **WCH CH569 as USB2 & USB3 Device Bulk (with burst)**



Firmware Reverse Engineering RISC-V USB3 Device Blob Ghidra USB3 Basic functions

Lot of “**basic/small**” functions to reverse
=> USB30_Device_Init(), USB30_switch_powermode(), USB30_device_setaddress()
=> USB30_IN_nump(), USB30_IN_set(), USB30_IN_clearIT(), USB30_send_ERDY()
=> USB30_OUT_set(), USB30_OUT_clearIT(), USB30_OUT_status()

Firmware Reverse Engineering RISC-V USB3 Device Blob Ghidra USB30_Device_Init



```
000000c4 13 07 00 14     li          a4,0x140
000000c8 98 c3           c.sw        a4,0x0(a5=>DAT_40008000)
000000ca 49 47           c.li        a4,0x12
000000cc d8 c3           c.sw        a4,0x4(a5=>DAT_40008004)
000000ce 37 57 4c 00     lui         a4,0x4c5
000000d2 13 07 17 b4     addi        a4,a4,-0x4bf
000000d6 b7 86 00 40     lui         a3,0x40008
000000da 11 a0           c.j         LAB_000000de

                     LAB_000000dc
000000dc 3d cb           c.beqz      a4,LAB_00000152

                     LAB_000000de
000000de 9c 4a           c.lw        a5,0x10(a3=>DAT_40008010)
000000e0 7d 17           c.addi      a4,-0x1
000000e2 91 8b           c.andi      a5,0x4
000000e4 e5 ff           c.bnez      a5,LAB_000000dc
000000e6 23 ae 06 06     sw          zero,0x7c(a3=>DAT_4000807c)
000000ea 23 ac 06 06     sw          zero,0x78(a3=>DAT_40008078)
000000ee 23 a6 06 08     sw          zero,0x8c(a3=>DAT_4000808c)
000000f2 23 a4 06 08     sw          zero,0x88(a3=>DAT_40008088)
000000f6 23 ae 06 08     sw          zero,0x9c(a3=>DAT_4000809c)
000000fa 23 ac 06 08     sw          zero,0x98(a3=>DAT_40008098)
000000fe 23 a6 06 0a     sw          zero,0xac(a3=>DAT_400080ac)
00000102 23 a4 06 0a     sw          zero,0xa8(a3=>DAT_400080a8)
00000106 23 ae 06 0a     sw          zero,0xbc(a3=>DAT_400080bc)
0000010a 23 ac 06 0a     sw          zero,0xb8(a3=>DAT_400080b8)
0000010e 23 a6 06 0c     sw          zero,0xcc(a3=>DAT_400080cc)
00000112 23 a4 06 0c     sw          zero,0xc8(a3=>DAT_400080c8)
00000116 23 ae 06 0c     sw          zero,0xdc(a3=>DAT_400080dc)
0000011a 23 ac 06 0c     sw          zero,0xd8(a3=>DAT_400080d8)
0000011e 23 a6 06 0e     sw          zero,0xec(a3=>DAT_400080ec)
00000122 23 a4 06 0e     sw          zero,0xe8(a3=>DAT_400080e8)
00000126 cd 47           c.li        a5,0x13
00000128 dc d2           c.sw        a5,0x24(a3=>DAT_40008024)
0000012a b7 07 03 00     lui         a5,0x30
0000012e 93 87 17 02     addi        a5,a5,0x21
00000132 9c d2           c.sw        a5,0x20(a3=>DAT_40008020)
00000134 23 a8 06 06     sw          zero,0x70(a3=>DAT_40008070)
00000138 9c 42           c.lw        a5,0x0(a3=>DAT_40008000)
0000013a 01 45           c.li        a0,0x0
0000013c 93 e7 27 00     ori         a5,a5,0x2
00000140 9c c2           c.sw        a5,0x0(a3=>DAT_40008000)
00000142 b7 c7 10 00     lui         a5,0x10c
00000146 93 87 d7 c7     addi        a5,a5,-0x383
0000014a 9c c6           c.sw        a5,0x8(a3=>DAT_40008008)
0000014c 89 47           c.li        a5,0x2
```

```
1
2  UINT8 USB30_Device_Init(void)
3
4  {
5    uint uVar1;
6    int iVar2;
7
8    gp = &__global_pointer$;
9    DAT_40008000 = 0x140;
10   DAT_40008004 = 0x12;
11   iVar2 = 0x4c4b41;
12   do {
13     uVar1 = DAT_40008010;
14     iVar2 = iVar2 + -1;
15     if ((uVar1 & 4) == 0) {
16       DAT_4000807c = 0;
17       DAT_40008078 = 0;
18       DAT_4000808c = 0;
19       DAT_40008088 = 0;
20       DAT_4000809c = 0;
21       DAT_40008098 = 0;
22       DAT_400080ac = 0;
23       DAT_400080a8 = 0;
24       DAT_400080bc = 0;
25       DAT_400080b8 = 0;
26       DAT_400080cc = 0;
27       DAT_400080c8 = 0;
28       DAT_400080dc = 0;
29       DAT_400080d8 = 0;
30       DAT_400080ec = 0;
31       DAT_400080e8 = 0;
32       DAT_40008024 = 0x13;
33       DAT_40008020 = 0x30021;
34       DAT_40008070 = 0;
35       uVar1 = DAT_40008000;
36       DAT_40008000 = uVar1 | 2;
37       DAT_40008008 = 0x10bc7d;
38       DAT_40008004 = 2;
39       return '\0';
40     }
41   } while (iVar2 != 0);
42   return '\x01';
43 }
44
```

Final C code with improvements

```
static int USB30_device_init(void)
{
    USBSS->LINK_CFG = 0x140;
    USBSS->LINK_CTRL = 0x12;
    uint32_t t = 0x4c4b41;
    while(USBSS->LINK_STATUS&4)
    {
        t--;
        if(t == 0)
            return -1;
    }
    for(int i = 0; i < 8; i++)
    {
        SS_TX_CONTRL(i) = 0;
        SS_RX_CONTRL(i) = 0;
    }
    USBSS->USB_STATUS = 0x13;

    USBSS->USB_CONTROL = 0x30021;
    USBSS->UEP_CFG = 0;

    USBSS->LINK_CFG |= 2;

    USBSS->LINK_INT_CTRL = 0x10bc7d;

    USBSS->LINK_CTRL = 2;
    return 0;
}
```

Firmware Reverse Engineering RISC-V USB3 Device Blob Ghidra USB30_IN_set

```
                    void __stdcall USB30_IN_Set(UINT8 endp, Funct
                      assume gp = 0x200008b0
      void            <VOID>          <RETURN>
      UINT8           a0:1            endp
      FunctionalState a1:4            lpf
      UINT8           a2:1            status
      UINT8           a3:1            nump
      UINT16          a4:2            TxLen
                    USB30_IN_Set

00000242 b7 87 00 40     lui         a5,0x40008
      assume gp = <UNKNOWN>
00000246 93 87 c7 07     addi        a5,a5,0x7c
0000024a 12 05           c.slli      endp,0x4
0000024c 3e 95           c.add       endp,a5
0000024e c2 06           c.slli      nump,0x10
00000250 6a 06           c.slli      status,0x1a
00000252 1c 41           c.lw        a5=>DAT_4000807c,0x0(endp)
00000254 d1 8e           c.or        nump,status
00000256 13 77 f7 7f     andi        TxLen,TxLen,0x7ff
0000025a d9 8e           c.or        nump,TxLen
0000025c f2 05           c.slli      lpf,0x1c
0000025e cd 8e           c.or        nump,lpf
00000260 dd 8e           c.or        nump,a5
00000262 14 c1           c.sw        nump,0x0(endp=>DAT_4000807c)
00000264 82 80           ret
```

```
1
2  void USB30_IN_Set(UINT8 endp,FunctionalState lpf,UINT8 status,UINT8 nump,UINT16 TxLen)
3
4  {
5    undefined3 in_register_00002029;
6    undefined3 in_register_00002035;
7
8    gp = &__global_pointer$;
9    (&DAT_4000807c)[CONCAT31(in_register_00002029,endp) * 4] =
10       CONCAT31(in_register_00002035,nump) << 0x10 | (uint)status << 0x1a | TxLen & 0x7ff |
11       lpf << 0x1c | (&DAT_4000807c)[CONCAT31(in_register_00002029,endp) * 4];
12   return;
13 }
14
```

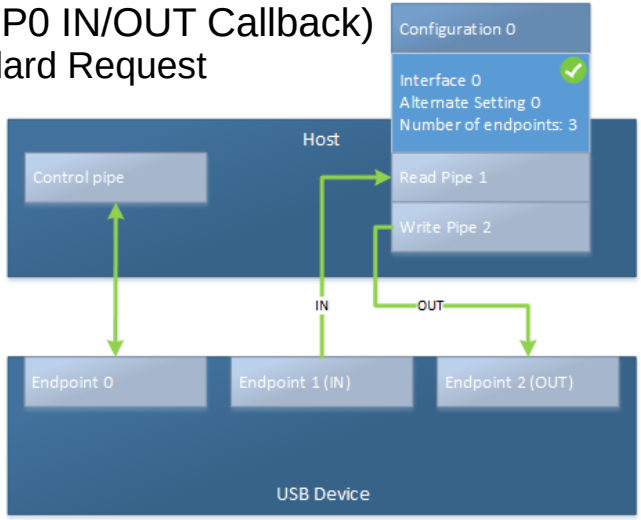
Final C code with improvements

```
static inline void USB30_IN_set(uint8_t endp,FunctionalState lpf,uint8_t status,uint8_t nump,uint16_t TxLen)
{
    vuint32_t* p = &USBSS->UEP0_TX_CTRL;
    p+= endp*4;
    *p = *p | (nump<<16) | (status<<26) | (TxLen & 0x7ff) | (lpf << 28);
}
```

Firmware Reverse Engineering RISC-V USB3 Device Blob Ghidra USB3 IRQHandler challenges

The Final Quest “**USB3 IRQ Handler Reverse Engineering**”

=> It is one of the most critical function, as it is the main USB3 device “state machine”
=> **Link Management Packet** (LMP)
=> Management of **End Points1 - 7** (EP IN & OUT Callback)
Specific to user’s application/requirements
=> Management of “**special**” **EndPoint 0** (EP0 IN/OUT Callback)
Used for USB3 Standard Request & Non Standard Request

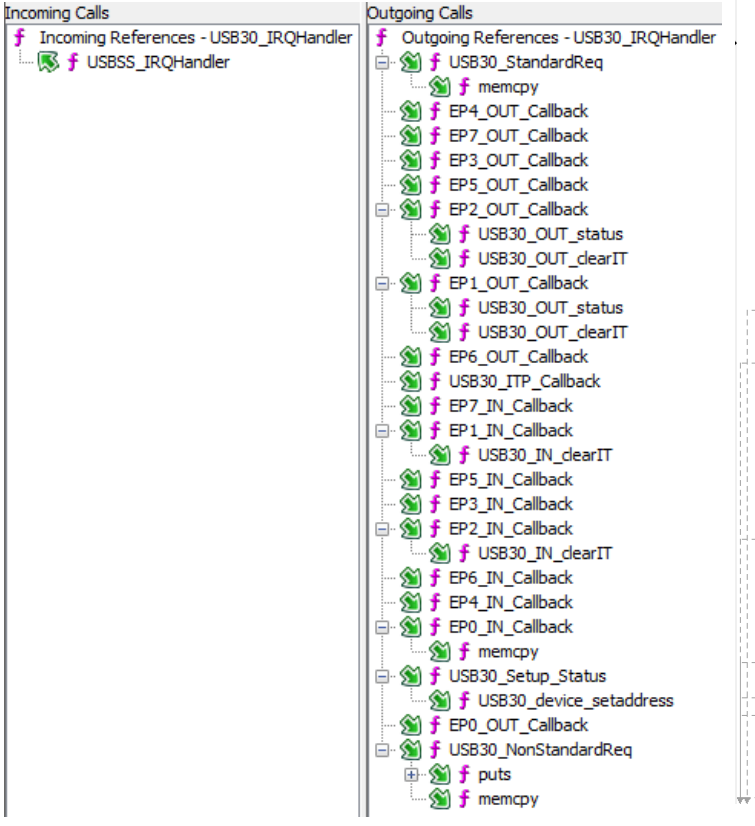


Firmware Reverse Engineering RISC-V USB3 Device Blob Ghidra USB3 IRQHandler details

- **One “complex” function** to reverse **USBSS_IRQHandler**() (calling USB30_IRQHandler())
- Main issue is that **big machine state “inline” some code from other functions** (mainly EP callback)
- **No details on registers** (no documentation => only register name / few defines)
- **Lot of research before to start** (reading the USB3 specifications …)
- **Hard to factorize all that “mess”**
  - It took lot of iterations rewriting/rebuilding(checking ASM code generated) multiple time until it “magically” works (at least doing USB3 enumeration)
- **Impossible to debug/trace USB3 in real-time**
  - UART traces are too slow (even at 5 Mbauds) and break USB3 enumeration …
  - HSPI or SerDes real-time DMA traces is a must have for that purpose but it was not ready
    - Was requiring USB3 to retrieve the traces => “**Chicken or the egg**” dilemma

Firmware Reverse Engineering RISC-V USB3 Device Blob Ghidra IRQHandler pseudo C code

Ghidra Function Call Trees



About 260 lines of pseudo C code (with unknown registers)

```
                        void __stdcall USB30_IRQHandler(void)
                        USB30_IRQHandler
0000032c 41 11          c.addi      sp,-0x10
     assume gp = <UNKNOWN>
0000032e 22 c4          c.swsp      s0,0x8(sp)
00000330 37 84 00 40    lui         s0,0x40008
00000334 5c 50          c.lw        a5,0x24(s0=>DAT_40008024)
00000336 06 c6          c.swsp      ra,0xc(sp)
00000338 85 8b          c.andi      a5,0x1
0000033a c1 c7          c.beqz      a5,LAB_000003c2
0000033c 5c 50          c.lw        a5,0x24(s0=>DAT_40008024)
0000033e 58 50          c.lw        a4,0x24(s0=>DAT_40008024)
00000340 a1 83          c.srli      a5,0x8
00000342 31 83          c.srli      a4,0xc
00000344 9d 8b          c.andi      a5,0x7
00000346 05 8b          c.andi      a4,0x1
00000348 b9 ef          c.bnez      a5,LAB_000003a6
0000034a 71 eb          c.bnez      a4,LAB_0000041e
0000034c 3c 5c          c.lw        a5,0x78(s0=>DAT_40008078)
0000034e 13 97 17 00    slli        a4,a5,0x1
00000352 63 5a 07 0e    bge         a4,zero,LAB_00000446
00000356 23 2c 04 06    sw          zero,0x78(s0=>DAT_40008078)
0000035a 78 58          c.lw        a4,0x74(s0=>DAT_40008074)
0000035c 74 58          c.lw        a3,0x74(s0=>DAT_40008074)
0000035e b7 07 02 20    lui         a5,0x20020
00000362 3e 97          c.add       a4,a5
00000364 b6 97          c.add       a5,a3
00000366 83 c7 07 00    lbu         a5,0x0(a5=>endp0RTbuff)
0000036a 03 44 07 00    lbu         s0,0x0(a4=>endp0RTbuff)
0000036e 93 f7 07 06    andi        a5,a5,0x60
00000372 63 9f 07 0e    bne         a5,zero,LAB_00000470
00000376 15 25          c.jal       USB30_StandardReq

                        LAB_00000378
00000378 62 04          c.slli      s0,0x18
0000037a 61 84          c.srai      s0,0x18
0000037c c1 67          c.lui       a5,0x10
0000037e 63 40 04 12    blt         s0,zero,LAB_0000049e
00000382 13 87 f7 ff    addi        a4,a5,-0x1
00000386 63 0e e5 12    beq         a0,a4,LAB_000004c2
0000038a 63 05 05 14    beq         a0,zero,LAB_000004d4
0000038e 13 07 00 20    li          a4,0x200
00000392 63 66 a7 04    bltu        a4,a0,LAB_000003de
00000396 37 87 00 40    lui         a4,0x40008
0000039a b7 06 01 04    lui         a3,0x4010
```

```
2  void USB30_IRQHandler(void)
3  {
4    UINT8 UVar1;
5    uint uVar2;
6    UINT32 ITPCounter;
7    int iVar3;
8    int iVar4;
9    UINT16 UVar5;
10   undefined2 extraout_var;
11   undefined2 extraout_var_00;
12   undefined2 extraout_var_01;
13   uint uVar6;
14   uint uVar7;
15   UINT8 UStack_14;
16   UINT8 UStack_13;
17   UINT16 aUStack_12 [5];
18
19   gp = &__global_pointer$;
20   uVar6 = DAT_40008024;
21   if ((uVar6 & 1) == 0) {
22     uVar6 = DAT_40008024;
23     if ((uVar6 & 2) != 0) {
24       uVar6 = DAT_40008058;
25       if ((uVar6 & 0x1e0) == 0xa0) {
26         DAT_40008048 = 0x2c0;
27         DAT_4000804c = 0;
28         DAT_40008050 = 0;
29       }
30       DAT_40008024 = 2;
         gp = &__global_pointer$;
         return;
       }
       uVar6 = DAT_40008024;
       if ((uVar6 & 8) == 0) {
         gp = &__global_pointer$;
         return;
       }
       ITPCounter = DAT_40008030;
       USB30_ITP_Callback(ITPCounter);
       DAT_40008024 = 8;
       gp = &__global_pointer$;
       return;
     }
     uVar6 = DAT_40008024;
     uVar2 = DAT_40008024;
     uVar7 = uVar6 >> 8 & 7;
     uVar6 = uVar2 >> 0xc & 1;
     if (uVar7 != 0) {
```

Firmware Reverse Engineering RISC-V USB3 Device Blob Ghidra IRQHandler identify block

Link Management Packet (LMP)
(managed by HW)

Setup Status => **Control Transfer Status Callback**
End Points 0 IN/OUT Callback

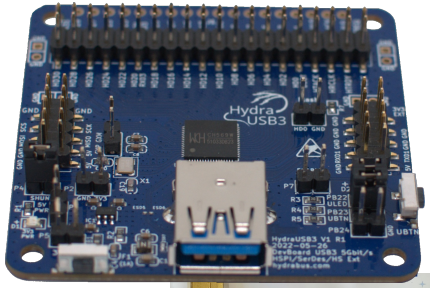
Standard Request (EP0) Callback
Non Standard Request (EP0) Callback

Send/Receive EP0 Data (IN/OUT)

EndPoints1 to 7 IN/OUT Callback

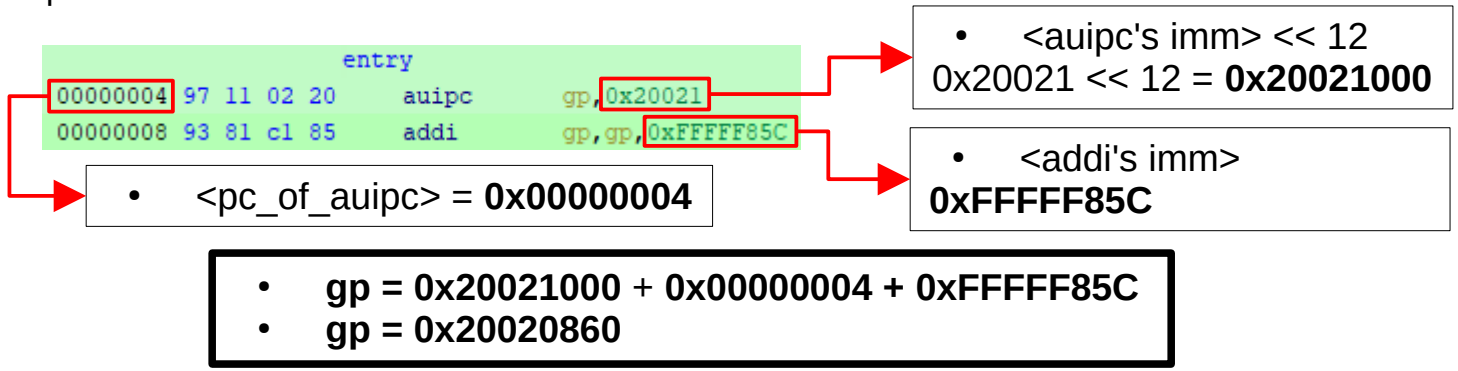
Firmware Reverse Engineering USB3 Device Firmware/Host example

- Reverse & rewrite in C the USB3 Device Bulk driver => DONE
  - USB2 & USB3 Device Bulk Driver
    https://github.com/hydrausb3/wch-ch56x-bsp/tree/main/usb/usb_devbulk
- Write an Example Firmware for USB2(HS)/USB3(SS) Device Bulk => DONE
  - Example Firmware USB3 Device Bulk Driver
    https://github.com/hydrausb3/hydrausb3_fw/tree/main/HydraUSB3_USB
- Write an Example Host Tool for USB2(HS)/USB3(SS) Device Bulk => DONE
  - Example Host Tool(USB3 Benchmark) to communicate with USB3 Device Bulk Driver
    https://github.com/hydrausb3/hydrausb3_host/tree/main/HydraUSB3_USB_benchmark
  - C open source multi-platform code using libusb
  - USB2 High Speed benchmark reach more than 48 MBytes/s average or more (depending on the PC)
  - USB3 Super Speed benchmark reach more than 330 MBytes/s average or more (depending on the PC)
    - 24% faster than original WCH USB3 CH372Device Demo which reach about 266 MBytes/s



Firmware Reverse Engineering Ghidra RISC-V RV32I/RV64I binary (firmware dump)

- RISC-V RV32I / RV64I use a global_pointer (gp) which is initialized in the entry code (and used in all functions)
  It is mandatory to initialize this gp register (as there is nothing automatic in Ghidra to do that)
- Very well explained in Ghidra GitHub Issue 2466
  https://github.com/NationalSecurityAgency/ghidra/issues/2466
  - The hint is to compute and set gp like described in the issue by mumbel
    - (<auipc's imm> << 12) + <pc_of_auipc> + <addi's imm>
- Example:



Conclusion

- Design of a Dev Board with WCH CH569 (HydraUSB3 v1) with KiCad 6
- Full reverse engineering (rewrite in C) of WCH CH569 SerDes
  - Firmware Blob reversed and rewritten to open source C
  - SerDes Physical & Data Link reversed (with the help of a fast Oscilloscope and open source tools like glscopeclient available on github)
- Full reverse engineering (rewrite in C) of WCH CH569 USB3 Device Bulk
  - Firmware Blob reversed and rewritten to open source C
  - Open source host tools (multi-platform) written in C using libusb
- Firmware Reverse Engineering Ghidra RISC-V RV32I/RV64I firmware dump hint about gp

Thanks to contributors on open source HydraUSB3 Firmware/Tools

- https://github.com/hydrausb3/hydrausb3_fw
  https://github.com/hydrausb3/wch-ch56x-isp

- Thanks to contributors
  - Hans Baier / Twitter @hansfbaier
  - Jules Maselbas / Twitter @jmsdok

- https://github.com/hydrausb3/wch-ch56x-bsp
  Thanks to Hans Baier / Twitter @hansfbaier

Contributors are welcome on https://github.com/hydrausb3

BONUS

Reverse Engineering WCH CH569 SerDes Data link (Idle sequence)

ALIGN Primitive Hex: **BC 4A 4A 7B**

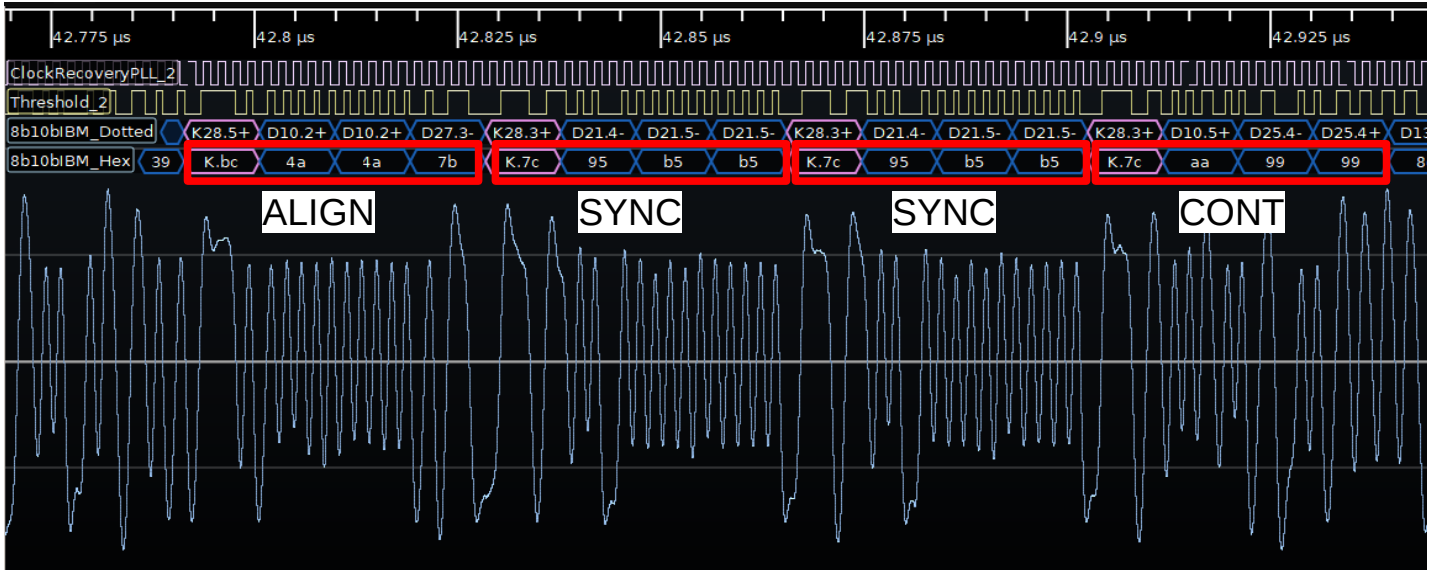
Align primitives establish dword(32bits) boundaries within the serial bit stream

2xSYNC Primitive Hex: **7C 95 B5 B5**

Used when in idle to maintain bit synchronization

CONT Primitive Hex: **7C AA 99 99**

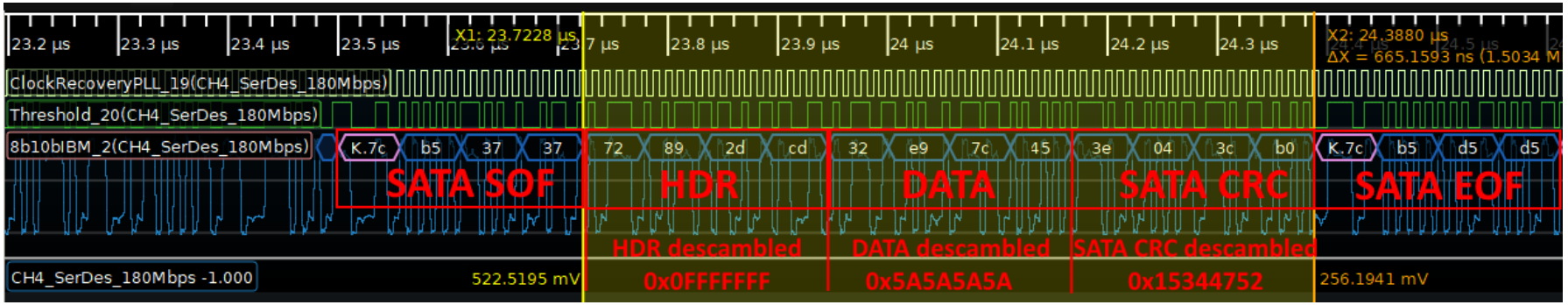
Allows long strings of repeated primitives to be eliminated



Reverse Engineering WCH CH569 SerDes Data link (data sequence)

1) **SATA SOF** 32bits => SOF Primitive Hex: **7C B5 37 37**
2) **HDR** (specific to WCH CH569W) 32bits
   => Not compliant with SATA => SATA expect Frame Info Structure(FIS)
   HDR Descrambled => Packet Number(4bits) + Packet Identifier(28bits) => Hex: **0F FF FF FF**
3) **DATA** (multiple of 32 bits) => SATA Descrambled Hex: **5A 5A 5A 5A**
4) **SATA CRC** 32bits => Compliant with SATA CRC32 => SATA Descrambled Hex: **15 34 47 52**
5) **SATA EO**F 32bits => EOF Primitive Hex: **7C B5 D5 D5**

Example WCH CH569 SerDes TX 180Mbps (Capture Rigol MSO5K/Partially decoded with glscopeclient)
Example with minimal payload 4 Bytes Hex: **5A 5A 5A 5A**



Reverse Engineering WCH CH569 SerDes more details

For more details see wch-ch569-serdes reverse-engineering
https://github.com/hydrausb3/wch-ch569-serdes