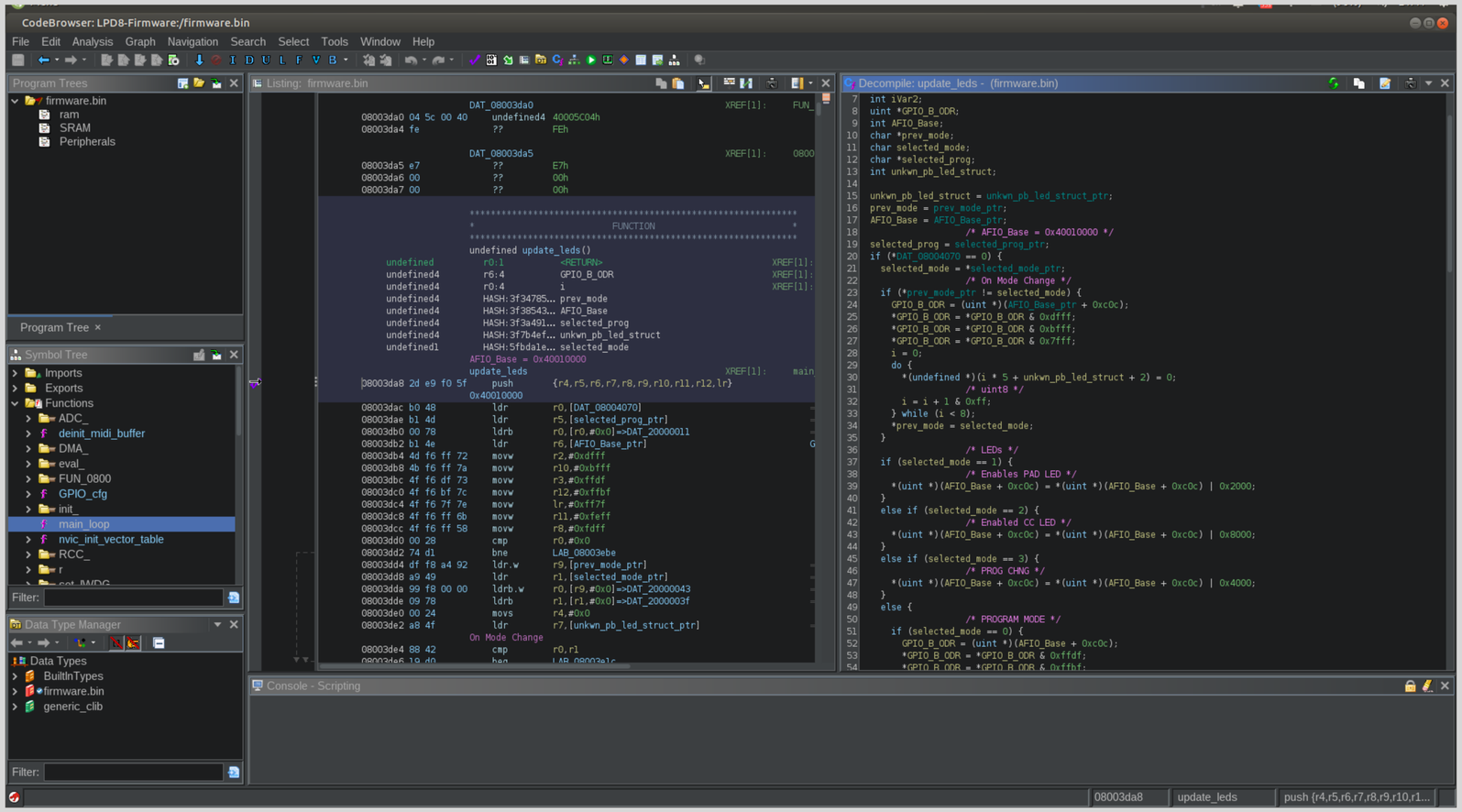
3. Reverse Engineering The Firmware

Reverse engineering the firmware is by far the most time-consuming task in this whole project. As of the time of writing, about one-third of the firmware code has been reverse-engineered. That said, this third covers arguably the most important and frequently used routines of the firmware (e.g., handling of pads, knobs, and LEDs). Ghidra is used to reverse engineer the firmware. Unfortunately, when reverse engineering embedded firmware, the code analysis lacks context, starting with minimal information. However, the bare-metal nature significantly reduces abstraction. This means that once a context is established, the reverse engineering efforts become significantly easier compared to, for instance, a multi-threaded application running on top of an OS. It's also worth mentioning that the ability to step through assembly instructions, inspect and alter memory, and modify register values via the SWD interface and GDB provided enormous help.



Currently Reverse-Engineered functions

```
SysTick_Handler
nvic_init_vector_table
SYSTICK_set_RELOAD
SYSTICK_set_TICKINT
SYSTICK_action
RCC_get_clock_freqs
init_SYSTICK
set_IWDG_KR
set_IWDG_PR
set_IWDG_RLR
init_IWDG
ADC_set_DMA
ADC_set_ADON
ADC_set_RSTCAL
ADC_read_RSTCAL
ADC_set_CAL
ADC_read_CAL
ADC_set_EXTTRIG_SWSTART
DMA_CCR_set_EN
DMA_clear_IFCR
DMA_read_ISR
DMA_set_IFCR
RCC_set_SW
RCC_set_USBPRE_BB
RCC_get_SWS
RCC_set_PLLON_BB
RCC_set_APB2ENR
RCC_set_AHBENR
ADC_set_SMPR_SQR
init_analog
GPIOs_cfg
write_midi_buffer
eval_knobs
read_analog
eval_pads
read_mode_pbs
eval_mode_pbs
rst_pads
update_leds
reload_IWDG
init_midi_buffer
deinit_midi_buffer
```