On-board LED blinking example

This application blinks an LED forever using the GPIO API. The board has an LED connected via a GPIO pin which is toggled. The LED must be configured using the ledo devicetree alias. This is usually done in the BOARD.dts file or a devicetree overlay. The same is

```
    leds {
        compatible = "gpio-leds";
        led0: led_0 {
            gpios = <&gpio0 13 GPIO_ACTIVE_LOW>;
            label = "Green LED 0";
        };
```

Figure 2: ledo is BOARD.dts

also present in the final devicetree source file after an application is built for a specific board. This can be found in **build/zephyr/zephyr.dts**

Once the application (blinking in this case) is built, zephyr stores the cache files in **/home/.cache/zephyr**. If the application is built again, west uses the cached files to reflect the output instead of building the whole application again. It is the speciality of zephyr that, in most cases, only specific parts of the code is compiled again where the newest changes are made, to save time (which is done using the cached contents). Here is an image showing the cached files: The

Figure 3: Cache

contents of **/home/.cache/zephyr** are not human-readable, which can be observed from the image above.

When an application is built, zephyr also generates some header files containing numerous **#define** statements. Some of the files are

- **/build/zephyr/include/generated/devicetree_unfixed.h**: This is a python-generated C++ header file (approximately 4200 lines of code) all describing several macros for a variety of nodes with different statuses.

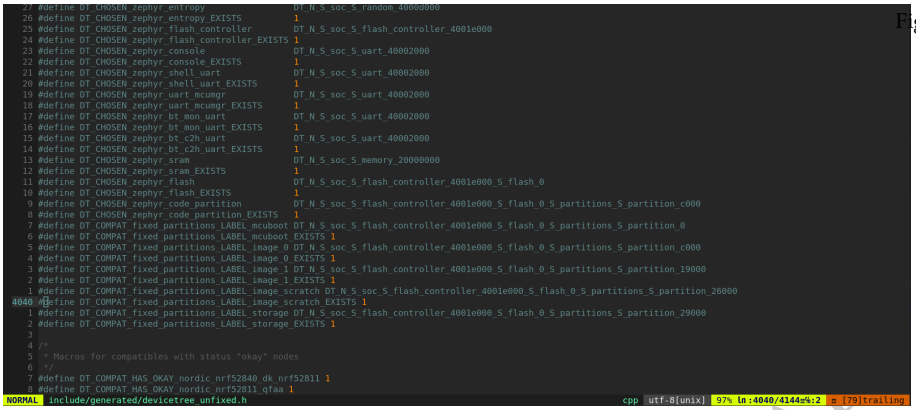

Figure 4: devicetree_unfixed.h

- **/build/zephyr/include/generated/device_extern.h**: This is also a C++ header file listing the device driver APIs used in other files (due to the fact that it employs **extern** keyword in it's declaration). Refer to device driver APIs and usage of extern const for more details.

Figure 5: device_extern.h

A configuration file is also loaded by zephyr in the build directory in the **/build/zephyr/.config**

```
CONFIG_GPIO=y
# CONFIG_SPI is not set
# CONFIG_I2C is not set
# CONFIG_KSCAN is not set
# CONFIG_WIFI is not set
CONFIG_NET_CONFIG_IEEE802154_DEV_NAME=""
# CONFIG_MODEM is not set
# CONFIG_UART_INTERRUPT_DRIVEN is not set
CONFIG_BOARD="nrf52840dk_nrf52811"
CONFIG_SOC="nRF52811_QFAA"
CONFIG_SOC_SERIES="nrf52"
CONFIG_NUM_IRQS=30
CONFIG_SYS_CLOCK_HW_CYCLES_PER_SEC=32768
# CONFIG_WATCHDOG is not set
CONFIG_HEAP_MEM_POOL_SIZE=0
CONFIG_ROM_START_OFFSET=0
# CONFIG_UART_NS16550 is not set
CONFIG_PM=y
# CONFIG_PM_DEVICE is not set
CONFIG_SOC_HAS_TIMING_FUNCTIONS=y
CONFIG_ARCH_HAS_CUSTOM_BUSY_WAIT=y
# CONFIG_CORTEX_M_SYSTICK is not set
CONFIG_CLOCK_CONTROL=y
CONFIG_NRF_RTC_TIMER=y
CONFIG_SYS_CLOCK_TICKS_PER_SEC=32768
CONFIG_BUILD_OUTPUT_HEX=y
CONFIG_FLASH_SIZE=192
CONFIG_FLASH_BASE_ADDRESS=0x0
# CONFIG_GPIO_MCUX is not set
# CONFIG_CPU_HAS_CUSTOM_FIXED_SOC_MPU_REGIONS is not set
# CONFIG_TINYCRYPT is not set
CONFIG_SERIAL=y
# CONFIG_UART_PL011 is not set
# CONFIG_SRAM_VECTOR_TABLE is not set
```

Figure 6: Caption

*Unsure explanation: The **prj.conf** file has similar configuration options as that of **.config*** file shown above. The **prj.conf** in the application directory may be a derivative of the **.config** file in some way.

*Alternate explanation: From the zephyr codebase, Kconfig and Kconfig.zephyr is parsed. All the board related configuration files are loaded from the zephyr codebase's **zephyr/boards/arm/nrf52840dk_nrf52811/nrf52840dk_nrf52811_defconfig** and then **west** merges the mentioned **defconfig** file with the application directory's **prj.conf** file. The final configuration file is saved to **/build/zephyr/.config**. If this is successful, the **Kconfig header** is generated in **/build/zephyr/include/generated/autoconf.h***

The image below is a comparison between the **.config** file and the **autoconf.h**

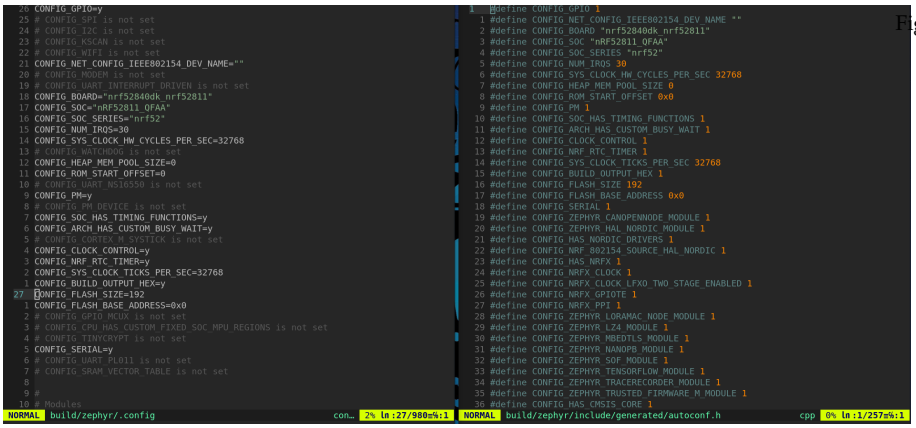

Figure 7: Caption