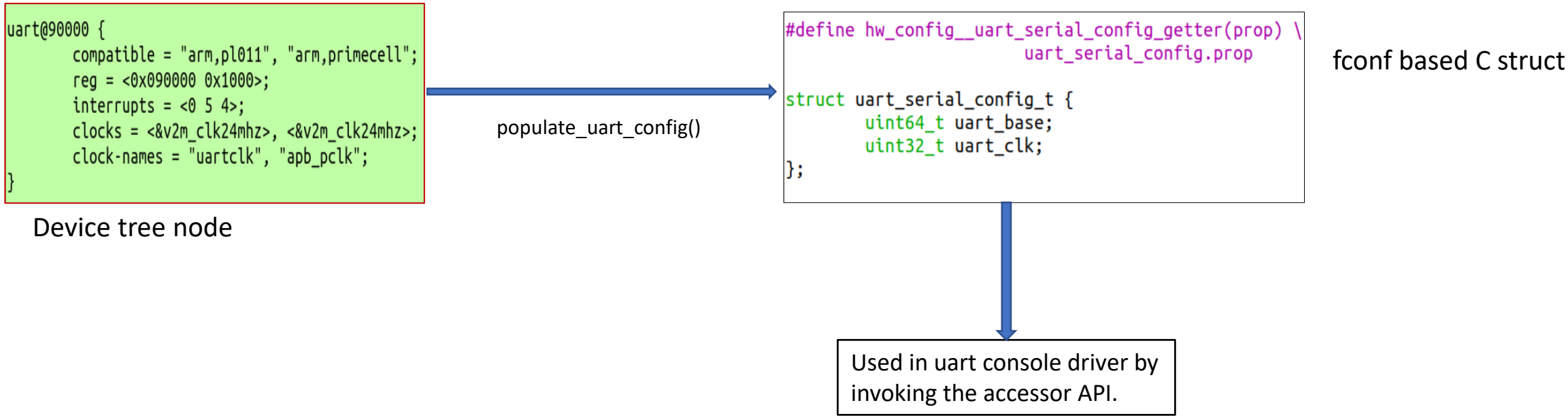
Firmware enhancements through fconf

Madhukar Pappireddy, Austin TX

May 2020

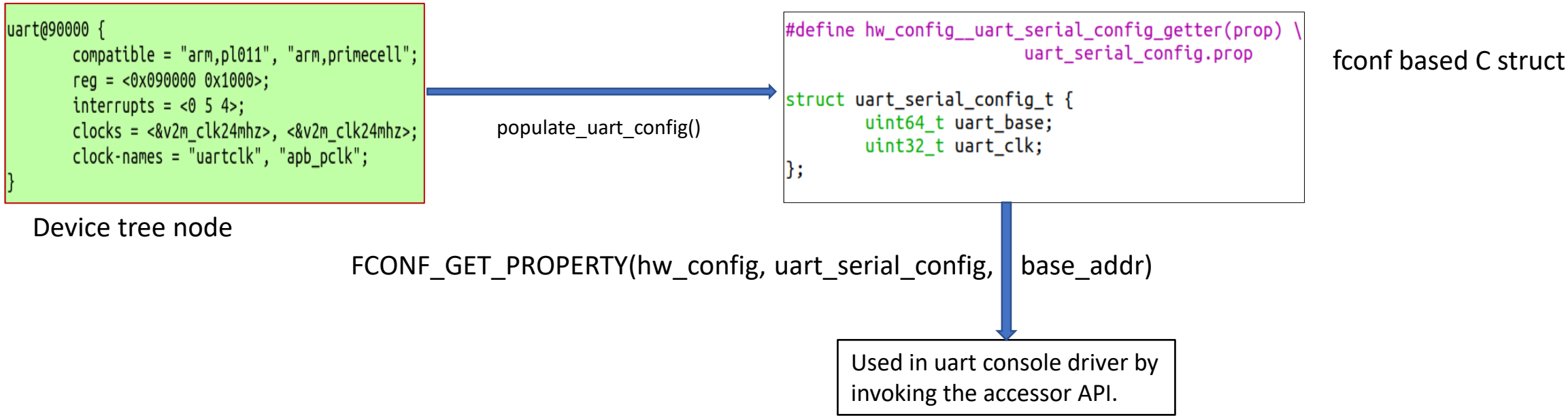
Brief introduction to fconf

- fconf is an abstraction layer for accessing platform specific data
- Requesting entity can query for a property without knowing what kind of backing store is used to hold the property.
- Properties are typically stored in C structures and are filled once using platform specific populate() callbacks.
- Ex: The populate function *populate_uart_config()* can retrieve the properties of uart node from dts and fill them in local C structures.



Brief introduction to fconf continued ..

- Property accessed using the API: FCONF_GET_PROPERTY(namespace, sub-namespace, prop)
- Preprocessor macro: translates to a pseudo function access :- namespace_sub-namespace_getter(prop)
- namespace loosely refers to a collection of group of properties: Ex: “hw_config”: represents configuration of various devices such as timer, uart, gic etc.
- sub-namespace loosely refer to group of related properties: Ex: “uart_serial_config” which holds properties of UART std-out serial device.



Brief introduction to fconf continued ..

- Each populate() function must be registered with the fconf framework
- Using FCONF_REGISTER_POPULATOR() macro.
  - Ex: FCONF_REGISTER_POPULATOR("HW_CONFIG", uart_config, populate_uart_config)
  - Tie populate() to config source (ex: dts)
- As part of platform setup, each BL invokes fconf_populate(config_type, config_dts)

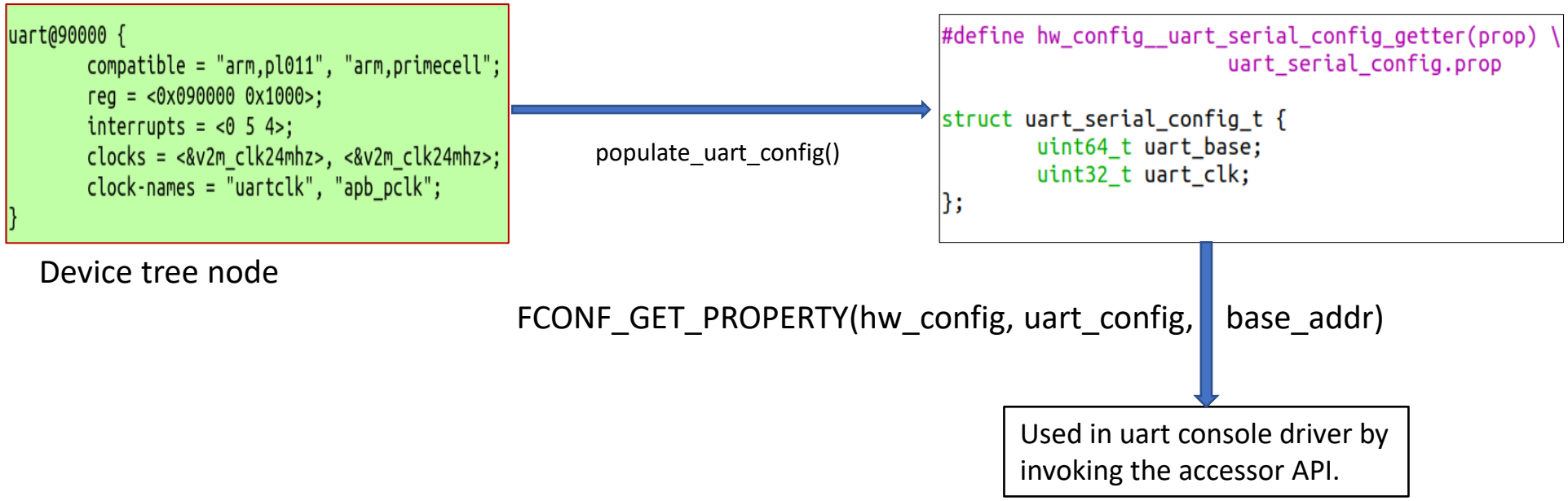
```
/* Go through all registered populate functions */
IMPORT_SYM(struct fconf_populator *, __FCONF_POPULATOR_START__, start);
IMPORT_SYM(struct fconf_populator *, __FCONF_POPULATOR_END__, end);
const struct fconf_populator *populator;

for (populator = start; populator != end; populator++) {
        assert((populator->info != NULL) && (populator->populate != NULL));

        if (strcmp(populator->config_type, config_type) == 0) {
                INFO("FCONF: Reading firmware configuration information for: %s\n", populator->info);
                if (populator->populate(config) != 0) {
                        /* TODO: handle property miss */
                        panic();
                }
        }
}
```

Leveraging fconf to move to dynamic configurations

- As presented in previous fconf session, we aim to enhance firmware by leveraging fconf framework.
- One such idea is to make various statically configured pieces of TF-A into dynamically configured components
- Example: IO Policies, Chain of Trust descriptors, hardware configuration of various devices such as UART, GIC etc.
- Instead of relying on hard coded compile time structures or macros, we intend to extract the platform specific configs using fconf and initialize various components in a BL image during runtime.
- Primary motivation :
  - Reduce the source code fragmentation within family of SoCs due to various platform specific definitions.
  - Evaluate the feasibility of having common TF-A BL images across multiple compatible platforms.

Leveraging fconf to move to dynamic configurations

- As a start, we move the platform specific properties and/or configurations to dts.
- Each platform can then provide the necessary populate() function , specific to its platform port
- Accessor APIs can then be integrated into a common library/device driver source code.
- This abstraction can help the shared library/driver code to be robust and re-usable across multiple platforms.



Making SDEI event description dynamic using fconf

- The platform specific SDEI event descriptors which specify properties of private and shared events such as event number, type of interrupt, events flags have been moved to device tree for FVP platform.

Static description in header file

```
/* ARM SDEI dynamic private event max count */
#define ARM_SDEI_DP_EVENT_MAX_CNT       3

/* ARM SDEI dynamic shared event max count */
#define ARM_SDEI_DS_EVENT_MAX_CNT       3
#else
/* ARM SDEI dynamic private event numbers */
#define ARM_SDEI_DP_EVENT_0             1000
#define ARM_SDEI_DP_EVENT_1             1001
#define ARM_SDEI_DP_EVENT_2             1002

/* ARM SDEI dynamic shared event numbers */
#define ARM_SDEI_DS_EVENT_0             2000
#define ARM_SDEI_DS_EVENT_1             2001
#define ARM_SDEI_DS_EVENT_2             2002

#define ARM_SDEI_PRIVATE_EVENTS \
        SDEI_DEFINE_EVENT_0(ARM_SDEI_SGI), \
        SDEI_PRIVATE_EVENT(ARM_SDEI_DP_EVENT_0, SDEI_DYN_IRQ, SDEI_MAPF_DYNAMIC), \
        SDEI_PRIVATE_EVENT(ARM_SDEI_DP_EVENT_1, SDEI_DYN_IRQ, SDEI_MAPF_DYNAMIC), \
        SDEI_PRIVATE_EVENT(ARM_SDEI_DP_EVENT_2, SDEI_DYN_IRQ, SDEI_MAPF_DYNAMIC)

#define ARM_SDEI_SHARED_EVENTS \
        SDEI_SHARED_EVENT(ARM_SDEI_DS_EVENT_0, SDEI_DYN_IRQ, SDEI_MAPF_DYNAMIC), \
        SDEI_SHARED_EVENT(ARM_SDEI_DS_EVENT_1, SDEI_DYN_IRQ, SDEI_MAPF_DYNAMIC), \
        SDEI_SHARED_EVENT(ARM_SDEI_DS_EVENT_2, SDEI_DYN_IRQ, SDEI_MAPF_DYNAMIC)
```

DTS node for SDEI

```
firmware {
        sdei {
                compatible = "arm,sdei-1.0";
                method = "smc";
                private_event_count = <3>;
                shared_event_count = <3>;
                /*
                 * Each event descriptor has typically 3 fields:
                 * 1. Event number
                 * 2. Interrupt number the event is bound to or
                 *    if event is dynamic, specified as SDEI_DYN_IRQ
                 * 3. Bit map of event flags
                 */
                private_events =        <1000 SDEI_DYN_IRQ SDEI_MAPF_DYNAMIC>,
                                        <1001 SDEI_DYN_IRQ SDEI_MAPF_DYNAMIC>,
                                        <1002 SDEI_DYN_IRQ SDEI_MAPF_DYNAMIC>;
                shared_events =         <2000 SDEI_DYN_IRQ SDEI_MAPF_DYNAMIC>,
                                        <2001 SDEI_DYN_IRQ SDEI_MAPF_DYNAMIC>,
                                        <2002 SDEI_DYN_IRQ SDEI_MAPF_DYNAMIC>;
        };
};
```

fconf_populate_sdei_dyn_config()

Populates SDEI C structures.

Properties accessed in arm SDEI driver code using fconf APIs.

Leveraging fconf to move to dynamic configurations

- We have chosen the FVP platform to test the feasibility of this effort.
- The following components have been identified to be made dynamic for FVP platform:
  - BL31 (runtime) UART configuration
  - Platform topology description
  - GICv3 configuration
  - Timer configuration
  - SDEI platform event descriptors
  - Chain of Trust descriptors
  - Platform IO policies
- These efforts are inline with the Total Compute vision of Arm to standardize software interfaces across platforms built on Arm Architecture.

Further improvements to fconf and firmware

- Investigating if there any other components that can be configured dynamically.
- Need inputs from TF-A community.
- Apart from this, we are also working on improving other aspects of TF-A using fconf framework such as
  - standardizing BL handoff arguments for accessing various firmware configuration files.
  - Implementing SETTER API in fconf framework to write to the platform properties.
  - Making fconf robust through necessary checks for mandatory and optional properties.
- Any major design decisions will be communicated through TF-A public mailing list and/or open forum meetings.

arm

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
תודה