1. 使用非类型模板特化代替 C 语言宏支持同一套底层驱动接口的多个实现。

modules/hal_nordic/nrfx/nrfx_config.h
sources/ble/init/init.c

```
  #if (BT_VER >= LL_VER_BT_CORE_SPEC_5_0)
    #ifdef INIT_CENTRAL
      LlExtScanMasterInit();
      LlExtInitMasterInit();
      LlPhyMasterInit();
      #if (BT_VER >= LL_VER_BT_CORE_SPEC_5_2)
        LlCisMasterInit();
        LlBisMasterInit();
        LlPowerControlInit();
      #endif
    #else
      #ifdef INIT_OBSERVER
        LlExtScanMasterInit();
        #if (BT_VER >= LL_VER_BT_CORE_SPEC_5_2)
          LlBisMasterInit();
        #endif
      #endif
    #endif
```

```
template <int BT_VER, int INIT_ROLE > struct BleInit;

struct DeviceVer50_Central {
  void init() {
      LlExtScanMasterInit();
      LlExtInitMasterInit();
      LlPhyMasterInit();
  }
}

strcut DeviceVer50_Observer {
  void init() {
      LlExtScanMasterInit();
  }
}

struct DeviceVer52_Central {
  void init() {
    DeviceVer50_Central::init();
    LlCisMasterInit();
    LlBisMasterInit();
    LlPowerControlInit();
  }
}

struct DeviceVer52_Observer {
  void init() {
    DeviceVer50_Observer::initObserver();
    LlBisMasterInit();
  }
}

template <> struct BleInitDriverChoice< 50 > {
  typedef DeviceVer50 type;
};

struct BleDriver {
  typedef BleInitDriverChoice< 50 >::type type;
};

BleDriver<50, Central>::type.init();
```

```
template<int PtrBitsVs32> struct DriverChoice;
template<> struct DriverChoice<-1> {
  // When bits/ptr < 32
  typedef SASDevice type;
};

template<> struct DriverChoice<0> {
  // When bits/ptr == 32
  typedef NASDevice type;
};

template<> struct DriverChoice<1> {
  // When bits/ptr > 32
  typedef BASDevice type;
};

struct Driver {
  enum { bitsPerVoidPtr = CHAR_BIT * sizeof(void*) };
  enum { ptrBitsVs32 = bitsPerVoidPtr >  32 ? 1:
                       bitsPerVoidPtr == 32 ? 0:
                                              -1
  };
  typedef DriverChoice<ptrBitsVs32>::type type;
};

int main()
{
  // ...;
  Driver::type d;
  d.doSomething();
}
```

- 减小命名空间污染
- 作用域
- 预处理器 vs 编译器

2. 类型安全

```
class GenericPtrStack {
protected:
  GenericPtrStack();
  ~GenericPtrStack();
  void push(void *object);
  void * pop();
private:
  ...     // same as before
};
```

```
template<typename T>
class Stack<T*>: private GenericPtrStack {
public:
  void push(T *objectPtr)
  { GenericPtrStack::push(objectPtr); }
  T * pop()
  { return static_cast<T*>(GenericPtrStack::pop()); }
};
```