Device-agnostic Firmware Execution is Possible: A Concolic Execution Approach for Peripheral Emulation

Chen Cao, Le Guan, Jiang Ming, Peng Liu
Recap: Emulator

• Software-based emulation techniques have demonstrated their pivotal roles in dynamically analyzing binary code.
Recap: Emulator

- **QEMU**, the most popular generic machine emulator, has built-in support for almost all popular processors.
Recap: Emulator

• **QEMU**, the most popular generic machine emulator, has built-in support for almost all popular processors.
Recap: Emulator

• **QEMU**, the most popular generic machine emulator, has built-in support for almost all popular processors.
Function Gap

• Gap 1: What to respond?
• Gap 2: When to request interrupt?
Motivating Observations

1. A large amount of peripheral accesses do not significantly influence firmware execution.

```c
    base->PCR[pin] = (base->PCR[pin] & ~PORT_PCR_MUX_MASK) | PORT_PCR_MUX(mux);
```
Motivating Observations

2. The rest of peripheral accesses that do actually influence firmware execution have an important effect on the execution path, and therefore, it is crucial to model them correctly.
Motivating Observations

3. If we can find a value that drives the execution along a correct path, then QEMU can usually execute the firmware as expected.

```c
void UART_WriteBlocking(UART_Type *base, const uint8_t *data, size_t length) {
    while (length--) {
        while (!((base->S1 & UART_S1_TDRE_MASK) ){})
            base->D = *(data++);
    }
}
```
void UART_WriteBlocking(UART_Type *base, const uint8_t *data, size_t length) {
    while (length--) {
        while (!((base->S1 & UART_S1_TDRE_MASK) )){}  
        base->D = *(data++);
    }
}
Laelaps

• Concolic = Concrete + Symbolic
Laelaps

• Context Preserving Scanning Algorithm
static void enet_init(...) {
    ...
    status = PHY_Init(...);
    if (kStatus_Success != status)
        LWIP_ASSERT("Cannot initialize PHY.", 0);
}

status_t PHY_Init(...) {
    ...
    result = PHY_Write(...);
    if (result == kStatus_Success) {
        result = PHY_Write(...);
        if (result == kStatus_Success) {
            /* Check auto negotiation complete. */
            while (counter--)
                result = PHY_Read(..., &bssReg);
            if (result == kStatus_Success) {
                PHY_Read(..., &ctlReg);
                if (((bssReg & ...) & (ctlReg & ...)) {
                    /* Wait a moment for Phy status stable. */
                    for (timeDelay = 0; timeDelay < PHY_TIMEOUT_COUNT; timeDelay++) {
                        /* Must be here to succeed. */
                        __ASM ("nop");
                    }
                }
                break;
            }
        }
    } else {
        if (!counter) {
            return kStatus_PHY_AutoNegotiateFail;
        }
    }
    return result;
}
static void enet_init(...) {
    ...
    status = PHY_Init(...);
    if (kStatus_Success != status)
        LWIP_ASSERT("\n\n Cannot initialize PHY.\n\n", 0);
}

status_t PHY_Init(...) {
    ...
    result = PHY_Write(...);
    if (result == kStatus_Success) {
        result = PHY_Write(...);
        if (result == kStatus_Success) {
            /* Check auto negotiation complete. */
            while (counter--) {
                result = PHY_Read(..., &bssReg);
                if (result == kStatus_Success) {
                    PHY_Read(..., &ctlReg);
                    if (((bssReg & ...) && (ctlReg & ...)) {
                        /* Wait a moment for Phy status stable. */
                        for (timeDelay = 0; timeDelay < PHY_TIMEOUT_COUNT; timeDelay++) {
                            /* Must be here to succeed. */
                            __ASM ("nop");
                        }
                        break;
                    }
                }
            }
            if (!counter) {
                return kStatus_PHY_AutoNegotiateFail;
            }
        }
    }
    /* Must be here to succeed. */
    __ASM ("nop");
}

return result;
static void enet_init(...) {
    ... 
    status = PHY_Init(...);
    if (kStatus_Success != status)
        LWIP_ASSERT("\n\nCannot initialize PHY.\n\n", 0);
}

status_t PHY_Init(...) {
    ... 
    result = PHY_Write(...);
    if (result == kStatus_Success) {
        result = PHY_Write(...);
        if (result == kStatus_Success) {
            /* Check auto negotiation complete. */
            while (counter--) {
                result = PHY_Read(..., &bssReg);
                if (result == kStatus_Success) {
                    PHY_Read(..., &ctlReg);
                    if (((bssReg & ...) && (ctlReg & ...)) {
                        /* Wait a moment for Phy status stable. */
                        for (timeDelay = 0; timeDelay < PHY_TIMEOUT_COUNT; timeDelay++) {
                            /* Must be here to succeed. */
                            __ASM ("nop");
                        }
                        break;
                    }
                }
                if (!counter) {
                    return kStatus_PHY_AutoNegotiateFail;
                }
            }
        }
        return result;
    }
}
static void enet_init(...) {
    ...
    status = PHY_Init(...);
    if (kStatus_Success != status)
        LWIP_ASSERT("\r\nCannot initialize PHY .\r\n", 0);
}

status_t PHY_Init(...) {
    ...
    result = PHY_Write(...);
    if (result == kStatus_Success) {
        result = PHY_Write(...);
        if (result == kStatus_Success) {
            /* Check auto negotiation complete . */
            while (counter--) {
                result = PHY_Read(..., & bssReg);
                if (result == kStatus_Success) {
                    PHY_Read(..., & ctlReg);
                    if ((( bssReg & ...) && (ctlReg & ...)) { /* Wait a moment for Phy status stable . */
                        for (timeDelay = 0; timeDelay < PHY_TIMEOUT_Count; timeDelay++) {
                            /* Must be here to succeed . */
                            __ASM ("nop");
                        }
                        break;
                    }
                    if (!counter) {
                        return kStatus_PHY_AutoNegotiateFail;
                    }
                }
                break;
            }
            return result;
        }
    }
    return result;
}
static void enet_init(...) {
    ...
    status = PHY_Init(...);
    if (kStatus_Success != status)
        LWIP_ASSERT("\r\nCannot initialize PHY.\r\n", 0);
}

status_t PHY_Init(...) {
    ...
    result = PHY_Write(...);
    if (result == kStatus_Success) {
        result = PHY_Write(...);
        if (result == kStatus_Success) {
            /* Check auto negotiation complete. */
            while (counter--){
                result = PHY_Read(..., &bssReg);
                if (result == kStatus_Success) {
                    PHY_Read(..., &ctlReg);
                    if (((bssReg & ...) && (ctlReg & ...)) { /* Wait a moment for Phy status stable. */
                        /* Must be here to succeed. */
                        __ASM ("nop");
                    } else {
                        break;
                    }
                } else {
                    return kStatus_PHY_AutoNegotiateFail;
                }
            }
        } else {
            return result;
        }
    } else {
        return result;
    }
}
Laelaps

• Heuristic #4: Fall-back Path.
  • Choose the one with the highest address.
Laelaps

• Interrupt Injecting
  • Periodically delivers activated interrupts.
Laelaps Implementation

- Concolic = Concrete + Symbolic
## Evaluation

- Firmware Emulation

<table>
<thead>
<tr>
<th>Board</th>
<th>RTOS</th>
<th>FW # w/o Human Intervention</th>
<th>FW # w/ Human Intervention</th>
<th>Failed #</th>
</tr>
</thead>
<tbody>
<tr>
<td>NXP FRDM-K66F</td>
<td>FreeRTOS, Bare-metal</td>
<td>14</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>NXP FRDM-KW41Z</td>
<td>FreeRTOS, Bare-metal</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>STM32100E_EVAL</td>
<td>FreeRTOS, Bare-metal</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>STM32 Nucleo-L152RE</td>
<td>ChibiOS, Mbed OS</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Evaluation

• Fidelity Analysis
Evaluation

• Fuzzing Mbed OS Firmware
  • Re-test the firmware without the real device.

• Analyzing FreeRTOS Firmware
Conclusion

• We distill the missing but essential parts for full system emulation of those devices.

• We fill the missing parts of full system device emulation by designing a symbolically-guided emulator.

• We demonstrate the potential of Laelaps by using it in combination with advanced dynamic analysis tools.
Thank you!