PES 5813 Final Project

Program Counter profiler

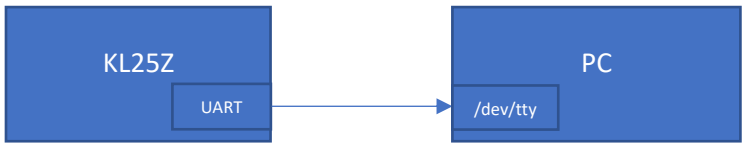
Hemanth Nandish

Develop a UART implementation to implement profiling, including writing scripts on PC side to interpret program counters.

Profiling a program involves logging the program counters periodically to help developers understand the consumption of CPU cycles in the code.

Functionality

1. The project consists of two parts: Host and Target. The target will gather the Program Counter information periodically and send it over UART to the Host. Host writes these logs to a file. This file can be parsed using a script to retrieve execution information.
2. The script needs to be aware of the functions and their corresponding address ranges. This needs to be obtained by parsing the readelf of the target binary.



Development

UART

On the target side, the code from Assignment 6 for UART serial communication was reused with slight modifications to allow only sending of data over UART and hence only Transfer Queue empty interrupt was enabled.

The UART rate is set 115200 bits/sec

Considering 8 data bits and 2 stops bits, the data bit rate is 0.8 * 115200 = 92160 bits/sec = 11520 bytes/sec

After reserving half of this bandwidth for PC samples and other half for DEBUG prints.

PC sample rate = 11520/2 = 5760 bytes/sec.

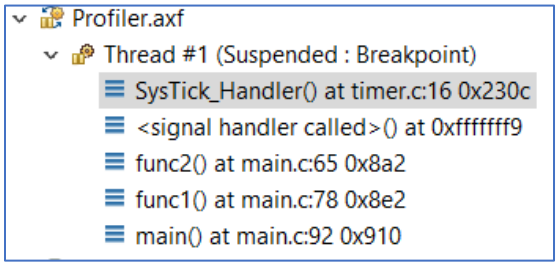
Considering 4 bytes for a 32bit address are to be sent at each interrupt, the Systick needs to trigger at 5760/4 = 1440 HZ

PC sampler

To sample the Program Counter value, I have used a Systick Interrupt to periodically sample the PC value.

The PC value is available on the stack during the Interrupt and can be obtained through pointer manipulation.

For example consider below example where the backtrace during the interrupt shows address executing before interrupt is inside **func2 at 0x8a2**



This address is available on the stack

```
0x20002fa0:     0x20002fc8      0xfffffff9      0x00000005      0x1ffff488
0x20002fb0:     0x000007da      0x000007d9      0x20002f4c      0x00001c03
0x20002fc0:     0x000008a2      0x01000000      0x00002430      0x000007da
0x20002fd0:     0x20002fd8      0x000008e3      0x00002438      0x000003e9
```

Then using some pointer manipulation like below

```
void SysTick_Handler()
{
    int var;
    int ptr;
    ptr=*(&var+11);
    char *buf = (char*)&ptr;
    int i=0;
    buf[i+3]=0xe3;
```

```
x/x buf
0x20002f90:     0x000008a2
```

The buf is pointer to a location containing the extracted PC value. I have added **0xE3** as the MSB which is needed during parsing the logs. This was chosen as 0xE3 is not an ASCII value.

This buf is the enqueue on the UART Write Queue.

Since this enqueue happens in interrupt context it gets exclusive access to the write queue and not affected by DEBUG prints of the program.

These logs can be stored in a file using PUTTY option to store logs.

Python Parser

1. Once these logs are available a python script can be used to parse through it to obtain the PC values.
2. Since the PC values are 32-bit, we read 4 bytes at a time and check if the MSB is 0xe3 and store the word.
3. Once we have a list of addresses, we can use the readelf utility to get a symbol table dump of the binary used on the KL25z to get the FUNCTION addresses. Below the column values with FUNC represent function with address in the previous column

```
232: 00002494     0 NOTYPE  GLOBAL DEFAULT    3 __exidx_end
233: 000000cc     0 NOTYPE  GLOBAL DEFAULT    1 __data_section_table_end
234: 00000197     8 FUNC    WEAK   DEFAULT    1 I2C1_IRQHandler
235: 000001b7     8 FUNC    WEAK   DEFAULT    1 UART1_IRQHandler
236: 00000411    52 FUNC    GLOBAL DEFAULT    1 init_fifo
237: 00001ce9   336 FUNC    GLOBAL DEFAULT    1 __flsbuf
238: 0000015f     8 FUNC    WEAK   DEFAULT    1 DMA2_IRQHandler
239: 0000014d     2 FUNC    WEAK   DEFAULT    1 TPM2_DriverIRQHandler
240: 20003000     0 NOTYPE  GLOBAL DEFAULT  ABS __top_SRAM
241: 000001d7     8 FUNC    WEAK   DEFAULT    1 TPM0_IRQHandler
242: 1ffff3c4     4 OBJECT  GLOBAL DEFAULT    3 errno
243: 00002494     0 NOTYPE  GLOBAL DEFAULT    1 _etext
244: 00000187     8 FUNC    WEAK   DEFAULT    1 LLWU_IRQHandler
245: 0000014d     2 FUNC    WEAK   DEFAULT    1 TSI0_DriverIRQHandler
246: 0000014d     2 FUNC    WEAK   DEFAULT    1 RTC_Seconds_DriverIRQHand
247: 0000021f     8 FUNC    WEAK   DEFAULT    1 TSI0_IRQHandler
248: 000006bd   116 FUNC    GLOBAL DEFAULT    1 cbfifo_length
249: 00000400     0 NOTYPE  GLOBAL DEFAULT  ABS _StackSize
250: 00000c35   112 FUNC    GLOBAL DEFAULT    1 UART0_IRQHandler
251: 00000133    16 FUNC    GLOBAL DEFAULT    1 bss_init
252: 0000014d     2 FUNC    WEAK   DEFAULT    1 LLWU_DriverIRQHandler
253: 00001c05    76 FUNC    GLOBAL DEFAULT    1 setvbuf
254: 000022e1    14 FUNC    GLOBAL DEFAULT    1 cbfifo_capacity
255: 1ffff47c     0 NOTYPE  GLOBAL DEFAULT    6 _noinit
256: 0000014d     2 FUNC    WEAK   DEFAULT    1 MCG_DriverIRQHandler
257: 00020000     0 NOTYPE  GLOBAL DEFAULT  ABS __top_PROGRAM_FLASH
258: 000001c7     8 FUNC    WEAK   DEFAULT    1 ADC0_IRQHandler
259: 00001f75   208 FUNC    GLOBAL DEFAULT    1 malloc
260: 00002375    18 FUNC    GLOBAL DEFAULT    1 remove
261: 00000207     8 FUNC    WEAK   DEFAULT    1 Reserved39_IRQHandler
```

4. The addresses of the functions are **sorted** in a list. A dictionary is used to represent the address as **key** and function name with count as the **value.**
5. Then the list is searched linearly to find the function address to which the sampled PC belongs to and the **hit count** for that function is **incremented**.
6. We can then sort the dictionary based on the hit values and get the **hit_value / total_samples** to get the percentage of execution occupied by the function.
7. The analysis below was calculated for the unoptimized PBKDF program from Assignment 5

```
ubuntu@ecen4133:/media/ext/Profiler$ python3 parser.py PBKDF2.axf putty8.log
PROFILING RESULT
function: ISHAInput                         37.69% samples:15070
function: ISHAProcessMessageBlock           32.61% samples:13039
function: hmac_isha                         16.91% samples:6763
function: ISHAPadMessage                    5.54% samples:2214
function: F                                 3.48% samples:1390
function: ISHAResult                        3.20% samples:1279
function: ISHAReset                         0.49% samples:195
function: cbfifo_capacity                   0.08% samples:32
function: cbfifo_length                     0.01% samples:2
TOTAL SAMPLES: 39984
```

DEMO

Consider a simple program with three function with different loop iterations and called sequentially.

```
void func3()
{

    int i=0;
    printf("func3");
    while(i++ < 1000000);
}


void func2()
{

    int i=0;
    printf("func2");
    while(i++ < 100000);

}

void func1()
{

    int i=0;
    printf("func1");
    while(i++ < 10000);
}
```

The result of profiling was as below. It reasonably matches our expectation that func3 > func2 > func1 in execution times.

```
PROFILING RESULT
function: func3                             86.57% samples:1379
function: func2                             11.42% samples:182
function: func1                             2.01% samples:32
TOTAL SAMPLES: 1593
```