James Rogan, 261157151. Fall 2025

ECSE 324 Lab 3 Report
Analysis of simulated resource utilization in an interupt-based embedded system (ARM-A9 DE1-SoC)

1 Introduction

An interactive rotating hex display was implemented in *ARMv7 assembly* using both polling and interrupts/timers and profiled on *CPUlator*. We profile the interrupt-based implementation to see what tasks the CPU spends most time on.

2 Methodology

To measure what fraction of time is spent on different activities, we use the total *instructions executed* as a proxy for elapsed time. While not perfectly accurate (since different instructions take different cycles on real hardware), the emulator uniformly reports instruction counts, making this a practical metric.

The methodology uses CPUlator's debugger breakpoints to establish instruction count checkpoints at key program locations:

From these checkpoints, we can calculate the number of instructions executed in each region over a sampling period.

The fraction of time spent on each activity is estimated using a fraction of specific instructions (between break points) over the total number of instructions.

3 Results

Table 1: Executed Instructions data for various break points

Breakpoint	**Executed instructions**
`IDLE` (before polling switches)	398
`IDLE` (after polling switches; switches changed)	402
`IDLE` (return from servicing switch change; before polling switches)	673
`SERVICE_IRQ`	674
`EXIT_IRQ`	691
`IDLE` (return from IRQ; before polling switches)	694
`IDLE` (after polling switches; no change)	698
`IDLE` (loop back; before polling switches)	775

Table 2: Instruction count deltas and computed time fractions

Category	**Instructions**	**Fraction**
Servicing interrupts (ISR + return)	20	5.3%
Polling slider switches	8	2.1%
IDLE/user code	349	92.6%
Total	**377**	**100%**

The ISR itself (SERVICE_IRQ to EXIT_IRQ) executed 17 instructions, with an additional 3 instructions for return overhead. Polling of slider switches consumed 4 instructions per execution, whether or not a switch change was detected. The bulk of execution time (92.6%) was spent in user code and IDLE loops, including 271 instructions dedicated to servicing the detected switch change event.

4 Discussion and Conclusion

This measurement reveals that the interrupt-driven approach allocates only 5.3% of processor time to interrupt servicing in two main loop iterations, demonstrating efficient ISR design. The ISR is kept minimal, deferring the actual switch change handling to user code running at normal priority. Polling overhead in this case is negligible at 2.1%, however has the disadvatage of executing at every loop iteration (while the ISR only executes when required by the user).

When extrapolated to continuous operation, a processor running this code indefinitely would spend far fewer of its executed instructions per iteration servicing interrupts, but always a constant amount polling switches, while the vast majority of time executing user and IDLE code. This distribution is favorable for responsive systems, as it minimizes interrupt latency impact while preserving processor availability for user tasks.