Energy Management in Mobile Systems

Lab 2: Encoding

Andrea Floridia - 224906, Robert Margelli - 224854

Contents

1. Purpose of the lab sessions

The purpose of the lab session was to familiarize with energy estimation in buses and data encoding. For this purpose several different data and address traces were produced and the energy required for their transmission was evaluated taking into consideration a simple wire model (i.e. approximating a wire to a capacitor). These traces were then encoded to understand how different encodings can provide energy savings.

2. Trace generation

The first tool we created was a trace generator by means of a C++ program (see `gen_stream.cpp, my_functions.cpp, my_functions.h` in the `ass_1` folder). This program can generate either data or address traces of various types according to the passed parameters. Our framework deals only with strings and not integer data types as we assessed that the former enabled an easier manipulation of bits in a stream.

In total, 19 traces of 1000 words each were generated: 9 for data and 10 for address streams.

For data:

1. 8-bit word with 2 random bits and 6 fixed (20% of bits are random);
2. 8-bit word with 4 random bits and 6 fixed (50% of bit are random);
3. 8-bit word with 6 random bits and 6 fixed (70% of bits are random);
4. 8-bit word with all random bits and none fixed;
5. 16-bit word with 3 random bits and 6 fixed (20% of bits are random);
6. 16-bit word with 8 random bits and 6 fixed (50% of bit are random);
7. 16-bit word with 11 random bits and 6 fixed (70% of bits are random);
8. 16-bit word with all random bits and none fixed;
9. 13-bit word with 7 random bits and 6 fixed (to demonstrate that our framework also worked for words with an odd number of bits).

For address:

1. 8-bit word with q=0 (completely random sequence of values);
2. 8-bit word with q=1 (completely sequential values);
3. 8-bit word with q=0.05 (sequential stream with a jump every 5 addresses);
4. 8-bit word with q=0.2 (sequential stream with a jump every 20 addresses);
5. 8-bit word with q=0.6 (sequential stream with a jump every 60 addresses);
6. 16-bit word with q=0 (completely random sequence of values);
7. 16-bit word with q=1 (completely sequential values);
8. 16-bit word with q=0.05 (sequential stream with a jump every 5 addresses);
9. 16-bit word with q=0.2 (sequential stream with a jump every 20 addresses);
10. 16-bit word with q=0.6 (sequential stream with a jump every 60 addresses);

Mainly, our intent was to have sets of streams that would provide us with differences in terms of energy estimation during the later experiments. For this reason, several configurations were created for both 8-bit and 16-bit words.

3. Bus encoding

The above streams were encoded with the `gen_encoded_streams.cpp` program which leverages on functions defined in `bus_encoder.cpp (ass_3` folder). Three core functions enable stream encodings, namely:

1. `bus_inv_enc()`: bus invert encoding;
2. `part_enc()`: two-partitioned bus invert encoding;
3. `gray_enc()`: gray encoding (used solely for address streams).

In total 48 encoded streams were generated and were later used to compare energy estimations.

4. Energy estimations

Energy estimation of streams was done according to the equation reported in the lab instructions and is implemented in `estimate_energy.cpp (ass_2 folder)`.

A bash script (`estimate_script.sh`) embraces all of the C++ files previously mentioned to perform in order:

1. trace generation;
2. trace encoding;
3. energy estimation of all traces.

The final output of the script are two text files: `no_encoding_results.txt` and `encoding_results.txt` which contain energy estimations of non-encoded traces and encoded ones respectively. These are the base of our final reasoning on bus encoding techniques. In the following we first analyze results for data traces (of both 8 and 16-bit streams) and then for address traces (still for 8 and 16-bit streams).

4.1 Energy estimation of data traces

- **without encoding:**

  Figure 1 shows the different energy consumptions in case of an 8-bit and 16-bit data stream. On the y-axis the energy is reported as the ratio between the energy consumption for the current value of m (E_m) and a reference case (m = 2 for the 8-bit data stream, m = 3 for the 16-bit data stream), whereas on x-axis the parameter m (random number of LSBs) is reported. The shapes of the two curves are quite similar, and in particular, the more bits change randomly, the higher is the energy required for transmitting the stream. The main reason for this behavior is that if more bits change in the pattern, parasitic capacitances of the wires will be charged and discharged more often yielding a higher energy consumption.

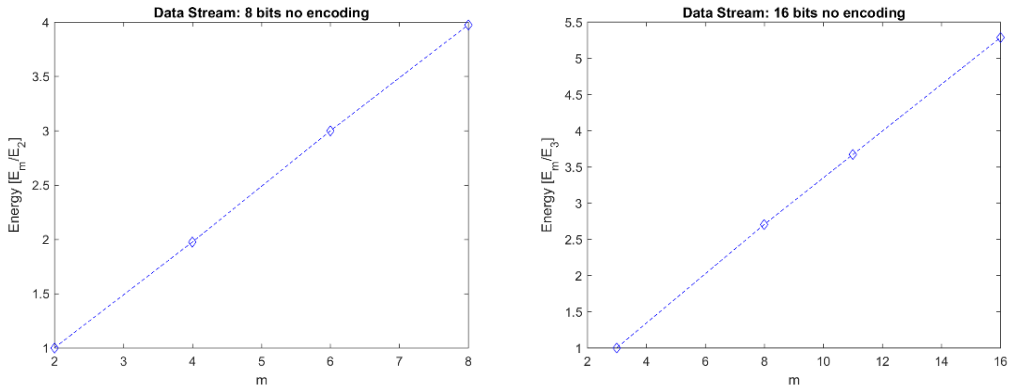

Figure 1- Energy of data streams without encoding.

- **With encoding:**

By encoding the streams, there are more opportunities to reduce the energy required by the system. Figure 2 shows that there are very few advantages in using bus invert as long as the number of bits that change random is less or equal to $n/2$. This is because bus invert performs better with pure random sequences, which is exactly the situation m = 8 (or m = 16 in the right-hand side plot). We notice that when n = 16, bus invert performs even worse. This is due to the distribution of Hamming distances, that seldom is greater than 8. Conversely, the two-partitioned bus invert always yields better results. Indeed, the energy consumptions are always below the cases of non-encoded and bus invert encoded stream. Two-partitioned bus invert encoding leverages the fact that basic bus invert performs better with short bit-width and pure random sequence. These characteristic are more likely to be met if only a sub-stream of the original one is considered.

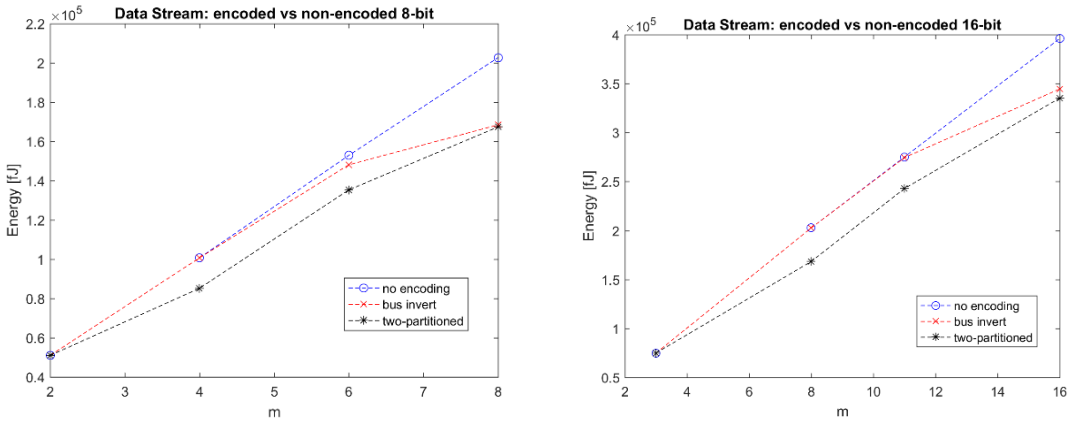

Figure 2 Comparison between encoded and non-encoded data streams.

4.2 Energy estimation of address traces

- **Without encoding:**

  The parameter q (x-axis) in both figures 3 and 4 represents the percentages of jumps in the addresses streams. On the y-axis there is the ratio between the considered value of q and the reference case (q = 0). The higher the bit-width, the better is for almost fully sequential sequence of addresses. This is because of very few bits change between an address and the subsequent (except for some rare cases). In practice, most of the bits composing an address seldom flip between two values in sequence.

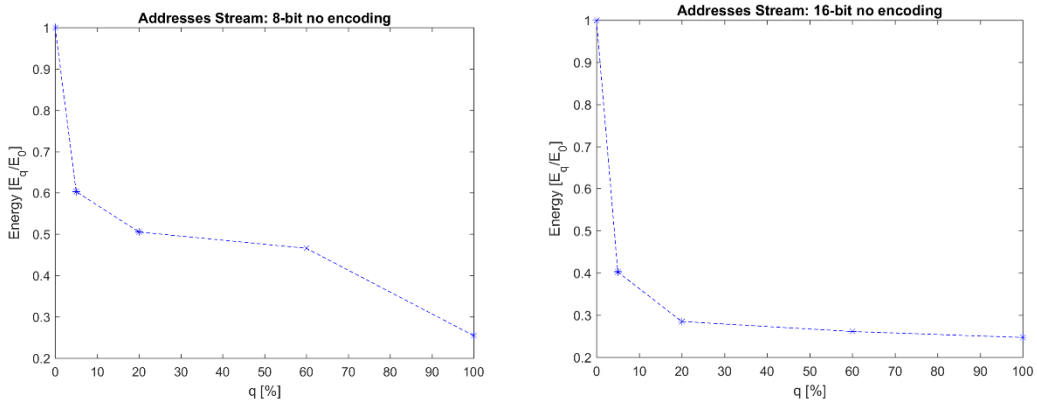

Figure 3 – Non-encoded address streams.

- **With encoding:**

The same reasoning done for data streams applies also for addresses using as encodings either bus invert or two-partitioned bus invert. As long as addresses are quite irregular (high number of jumps) bus invert based encodings result advantageous (figure 4), but once the number of jumps in the stream decreases, Gray encoding always yields better results than any other encodings.

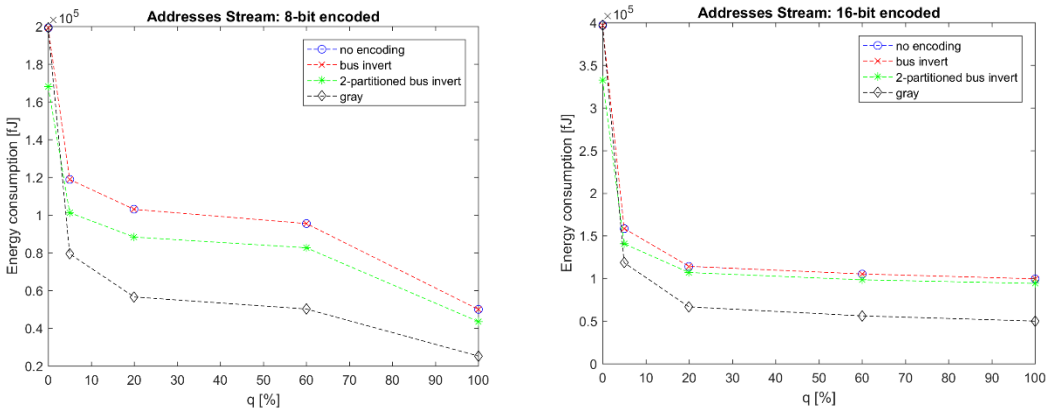

Figure 4 - Comparing encoded and non-encoded addresses streams.

4.3 Energy estimation odd data stream

For sake of completeness, we here report the energy consumption for the odd bit-width data stream:

Without Encoding	179050 fJ
Bus Invert	179050 fJ
Two-partitioned	145650 fJ

Table 1 – Odd bit-width data stream energy consumption

As expected, bus invert does not yield any advantages since it is very unlikely that more than 6 bits change their value between two consecutive words. The best strategy is again to apply the two-partitioned bus invert encoding.

5. Conclusions

The developed framework gave us the means to experiment with different streams and study the performance of encodings.

It is clear that if dealing with long bit-width data streams (not fully random), the best strategy is two-partitioned bus invert. On the other hand, for short bit-width random sequence bus invert is the best choice either for the performance (better energy savings that any other) and for the reduced overhead (1 additional wire and simple logic to be implemented).

For what concerns addresses, Gray encoding is always the best choice unless extremely irregular programs have to be executed (which is in practice very unlikely to occur).