Report Number: WUCSE-2004-39

2004-07-06

CMOS Implementations of a Range Check Circuit

Edward W. Spitznagel

TCAMs are the most popular practical approach to high performance packet classifica-tion, but they suffer from inefficient handling of range matches; the standard approach of rule replication can result in a 2-6x increase in TCAM words needed, for typical firewall databases. We describe three CMOS implementations of a range check circuit to address this problem; the most efficient of these designs allows classification on the standard IPv4 5-tuple with only a 46% increase in transistor count, rather than relying on rule replication. By avoiding replication, the overall transistor count required is only 24% to 78% of the stan-dard TCAM... **Read complete abstract on page 2.**

CMOS Implementations of a Range Check Circuit

Edward W. Spitznagel

Complete Abstract:

TCAMs are the most popular practical approach to high performance packet classifica-tion, but they suffer from inefficient handling of range matches; the standard approach of rule replication can result in a 2-6x increase in TCAM words needed, for typical firewall databases. We describe three CMOS implementations of a range check circuit to address this problem; the most efficient of these designs allows classification on the standard IPv4 5-tuple with only a 46% increase in transistor count, rather than relying on rule replication. By avoiding replication, the overall transistor count required is only 24% to 78% of the stan-dard TCAM design, for real filter databases used in this study; power dissipation is reduced similarly. Also, range check support greatly simplifies creation and maintenance of the TCAM contents, since there is now a one-to-one correspondence between filters and TCAM entries. Additionally, we show how to construct a more versatile device using range-check sub-fields that can be chained together as needed.

Department of Computer Science & Engineering

Washington University in St. Louis
School of Engineering & Applied Science

2004-39

CMOS Implementations of a Range Check Circuit

Authors: Spirznagel; Edward W.

July 6, 2004

Abstract: TCAMs are the most popular practical approach to high performance packet classification, but they suffer from inefficient handling of range matches; the standard approach of rule replication can result in a 2-6x increase in TCAM words needed, for typical firewall databases. We describe three CMOS implementations of a range check circuit to address this problem; the most efficient of these designs allows classification on the standard IPv4 5-tuple with only a 46% increase in transistor count, rather than relying on rule replication. By avoiding replication, the overall transistor count required is only 24% to 78% of the standard TCAM design, for real filter databases used in this study; power dissipation is reduced similarly. Also, range check support greatly simplifies creation and maintenance of the TCAM contents, since there is now a one-to-one correspondence between filters and TCAM entries. Additionally, we show how to construct a more versatile device using range-check sub-fields that can be chained together as needed.

CMOS Implementations of a Range Check Circuit

Edward W. Spitznagel

WUCSE-2004-39

July 6, 2004

Department of Computer Science and Engineering
Campus Box 1045
Washington University
One Brookings Drive
St. Louis, MO 63130-4899

Abstract

TCAMs are the most popular practical approach to high performance packet classification, but they suffer from inefficient handling of range matches; the standard approach of rule replication can result in a 2-6x increase in TCAM words needed, for typical firewall databases. We describe three CMOS implementations of a range check circuit to address this problem; the most efficient of these designs allows classification on the standard IPv4 5-tuple with only a 46% increase in transistor count, rather than relying on rule replication. By avoiding replication, the overall transistor count required is only 24% to 78% of the standard TCAM design, for real filter databases used in this study; power dissipation is reduced similarly. Also, range check support greatly simplifies creation and maintenance of the TCAM contents, since there is now a one-to-one correspondence between filters and TCAM entries. Additionally, we show how to construct a more versatile device using range-check sub-fields that can be chained together as needed.

CMOS Implementations of a Range Check Circuit

Edward W. Spitznagel
ews1@wustl.edu
Washington University
St. Louis, MO 63130-4899

1. Introduction

High performance packet classification is cruicial to a variety of new networking services. A number of algorithmic techniques have been proposed [4], but TCAMs are still the most popular practical approach, due to their deterministic high performance and simplicity of management. TCAMs, however, suffer from inefficient handling of range matches. The standard technique involves representing each range as a set of prefixes, replicating rules as needed. For filter databases studied in [1], this results in the ruleset increasing by a factor of two to six in size. In this report, we describe range check circuits which avoid the need for rule replication. The most efficient of these designs allows classification on the standard IPv4 5-tuple with 46% increase in transistor count, by using range check hardware for the port fields. Also, since there is now a one-to-one correspondence between filters and TCAM entries, the range check support greatly simplifies creation and maintenance of the TCAM contents.

This is accomplished by using standard TCAM logic for the source and destination IP addresses, transport protocol number, and transport protocol flags, and range match logic for the transport layer port numbers. E.g. each Extended TCAM [1] word might have 88 bits of standard TCAM logic, and two 16-bit wide range match fields. Using 44 transistors per bit of range match, and 16 per bit of standard TCAM matching, this comes out to 2816 transistors per word vs. a standard TCAM's 1920 transistors per word.

It is also worth noting that range matching can also be applied to other fields in certain cases. For example, when using the third P^2C encoding scheme described in [5], efficient range match hardware would eliminate the need for storing multiple ternary match conditions per rule (i.e. rule replication), provided that the bits for each field are allocated in a contiguous fashion.

The implementations described in this report operate by storing the lower and upper bound for each range match, and using dedicated range check circuitry that performs the comparison in a set of stages. The difference in each design lies in how the range check sub-circuit is implemented.

Section 2 describes the most straightforward scheme, which uses four inter-stage signals. A design using three inter-stage signals is given in Section 3; a more efficient refinement of this design is described in Section 4. Finally, in Section 5 we show how to construct a more versatile device using range-check sub-fields that can be chained together as needed.

2. Implementation with four inter-stage signals

The first circuit presented is the most straightforward of the three designs. This circuit consists of a separaste stage for each bit, and the comparison proceeds from the most significant bits to the least significant bits.

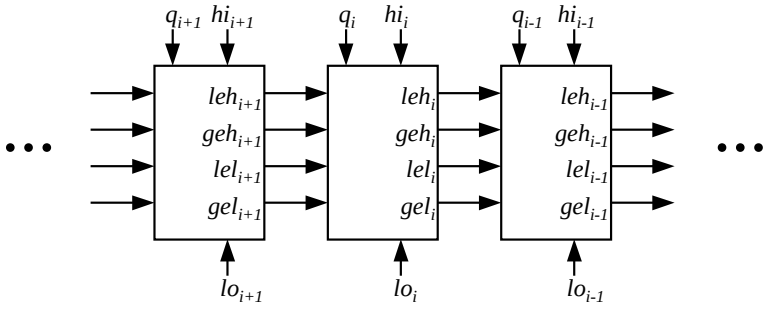

Figure 1: Iterative structure of range check circuit

The iterative structure of the circuit is shown in Figure 1, where hi_i, lo_i, and q_i represent bit i of the stored upper bound, the stored lower bound, and the query value, respectively. The four inter-stage signals geh_i, leh_i, gel_i, and lel_i are used to represent the following, where W is the width of a word:

geh_i: The quantity represented by the first $W-i$ bits of the query (i.e. q_{W-1} through q_i) is greater than or equal to the quantity represented by the first $W-i$ bits of the stored upper bound hi.

leh_i: The quantity represented by the first $W-i$ bits of the query is less than or equal to the quantity represented by the first $W-i$ bits of the stored upper bound hi.

gel_i: The quantity represented by the first $W-i$ bits of the query is greater than or equal to the quantity represented by the first $W-i$ bits of the stored lower bound lo.

lel_i: The quantity represented by the first $W-i$ bits of the query is less than or equal to the quantity represented by the first $W-i$ bits of the stored lower bound lo.

The names are abbreviations indicating that, up to bit i, the query is **g**reater (**l**ess) than or **e**qual to **h**i (**l**o).

The signals geh_W, leh_W, gel_W, and lel_W, which are inputs to the first stage, are always asserted. At that point, we can think of it as having compared the first zero digits (i.e. an empty string) from the query against the first zero digits of the upper and lower bounds (also empty strings.) The empty strings are equal; therefore those four signals are asserted.

The assertion of both leh_0 and gel_0 at the same time happens if and only if the query value q is within the range defined by hi and lo, inclusive; therefore, a "query is in range" singal can be formed by taking the logical AND of signals leh_0 and gel_0.

The logic for the upper bound check of one stage is shown in Figure 2. If the query bits before this stage are greater than hi (i.e. leh_{i+1} is not asserted), then the query value is still greater than hi once this stage is included as well; therefore we ensure in this case that leh_i is not asserted. If, on the other hand, the query bits before this stage are not greater than hi, then there is only one condition under which the query value including this bit can be greater than hi. That condition is

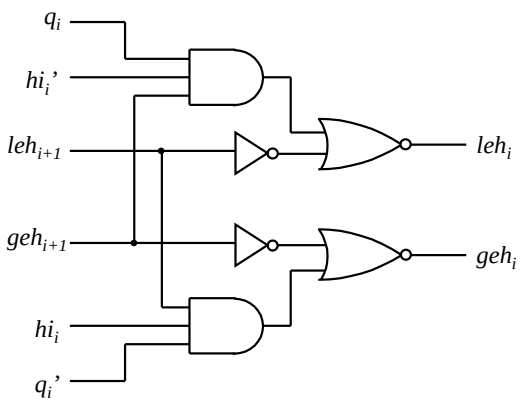

Figure 2: Range-check sub-circuit for upper bound comparison

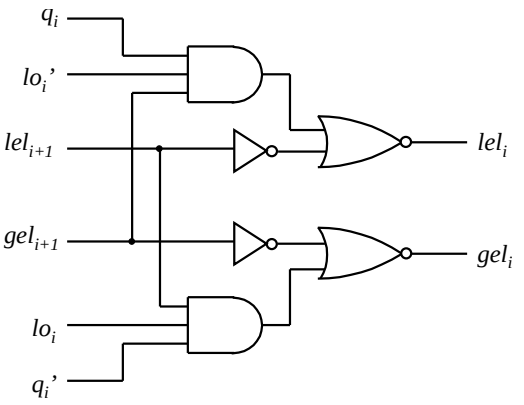

Figure 3: Range-check sub-circuit for lower bound comparison

that the previous query bits equal hi (implied by geh_{i+1} and leh'_{i+1}, but we can omit leh'_{i+1} since that case already causes us to assert leh'_i), q_i is 1, and hi_i is 0; therefore in this other case we also ensure leh_i is not asserted. In all other cases we assert leh_i.

Each AND gate and OR gate pairing in the sub-circuit can be implemented as a compound gate using 8 transistors. Each inverter requires 2 transistors. Thus this part of the sub-circuit requires 20 transistors.

The logic for lower bound check is shown in Figure 3; its operation is the same as the upper bound check, except that it uses lo_i (bit ith of the lower bound) instead of hi_i (bit i of the upper bound.) This part of the sub-circuit also requires 20 transistors.

Each stage requires an upper bound check (20 transistors), a lower bound check (20 transistors), and two SRAM storage cells (6 transistors each) for storing hi_i and lo_i. Thus 52 transistors are requires for each bit in the query, using this design. The bounds checking logic for the first stage can actually be simplified somewhat, since geh_W, leh_W, gel_W, and lel_W are always asserted. Similarly, the final stage does not need to generate the signals geh_0 and lel_0. This can reduce the transistor count of those particular stages, but the middle stages still need 52 transistors each.

3. Implementation with three inter-stage signals

Using three inter-stage signals instead of four can allow us to reduce the transitor count, if we are sufficiently careful. This circuit also consists of a separate stage for each bit, similar to the previous circuit. The overall structure of the new circuit is shown in Figure 4.

The three inter-stage signals are:

eh_i: The quantity represented by the first $W - i$ bits of the query (i.e. q_{W-1} through q_i) is equal to the quantity represented by the first $W - i$ bits of the stored upper bound hi.

el_i: The quantity represented by the first $W-i$ bits of the query is equal to the quantity represented by the first $W - i$ bits of the stored lower bound lo.

oor_i: The quantity represented by the first $W - i$ bits of the query is out of range (i.e. is above hi or below lo.)

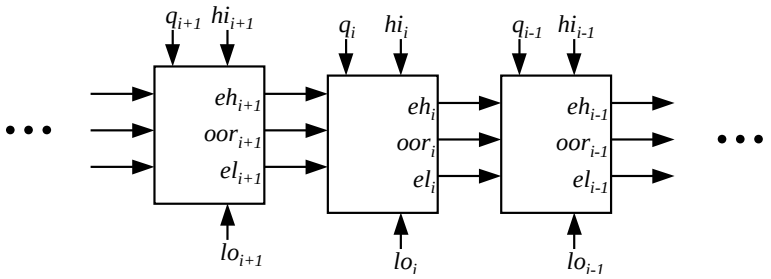

Figure 4: Iterative structure of range check circuit

The names are abbreviations indicating that, up to bit i, the query is **e**qual to **h**i, **e**qual to **l**o, or **o**ut **o**f **r**ange.

Logic expressions for each inter-stage signal can be written as follows:

$$eh_i \equiv eh_{i+1} \wedge (q_i = hi_i)$$

$$el_i \equiv el_{i+1} \wedge (q_i = lo_i)$$

$$oor_i \equiv oor_{i+1} \vee (eh_{i+1} \wedge hi_i' \wedge q_i) \vee (el_{i+1} \wedge lo_i \wedge q_i')$$

And we use as initial conditions that eh_W and el_W are asserted, and oor_W is not.

The final answer is determined simply by looking at the value of oor_0. This signal is asserted if and only if the query is out of range.

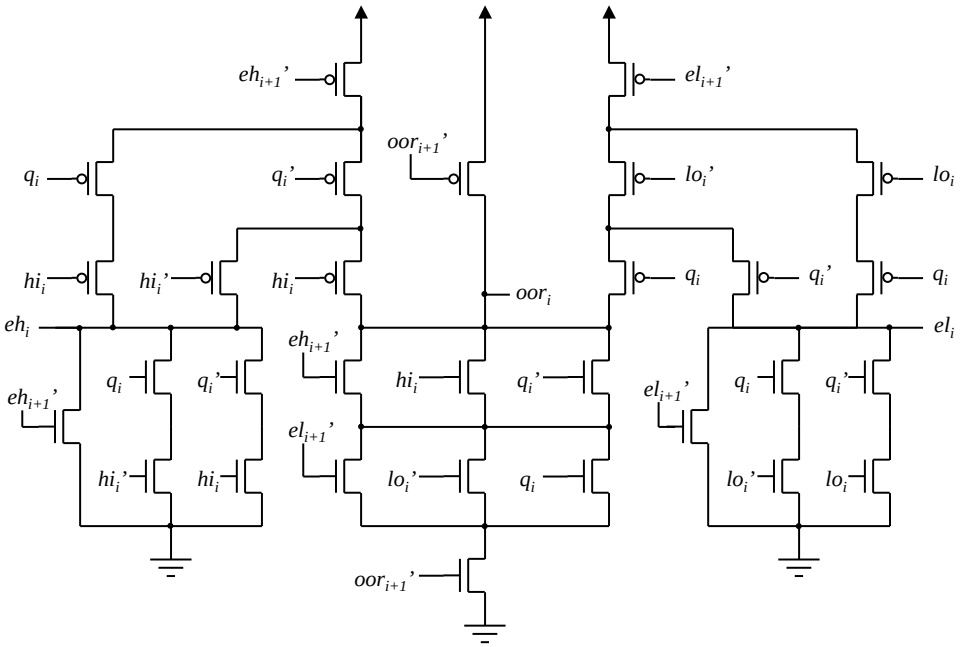

Figure 5: Range-check sub-circuit using three inter-stage signals

A circuit for generating the inter-stage signals is shown in Figure 5. This is the result of pulling out common subexpressions in the circuit, after rewriting the logic expressions as follows:

$$eh_i \equiv eh_{i+1} \wedge (q_i = hi_i) \equiv eh_{i+1} \wedge ((q_i \wedge hi_i) \vee (q_i' \wedge hi_i'))$$

$$el_i \equiv el_{i+1} \wedge (q_i = lo_i) \equiv el_{i+1} \wedge ((q_i \wedge lo_i) \vee (q_i' \wedge lo_i'))$$

Thus, this subcircuit requires only 36 transistors, including 6 for the inverters to generate eh'_{i+1}, el'_{i+1}, and oor'_{i+1}. Adding in the 12 transistors for the two SRAM storage cells (for storing hi_i and lo_i) brings the total to 48 transistors per bit. As before, the first stage circuitry can be somewhat simplified, given that the values of eh_W, el_W, and oor_W are fixed. And the last stage can be simplified, because the signals eh_0 and el_0 need not be generated.

4. Revised implementation with three inter-stage signals

We can reduce the transistor count further by relaxing the constraints used to define two of the inter-stage signals in the previous circuit. The top-level structure of the new circuit, shown in Figure 6, is the same except for the labels of the inter-stage signals.

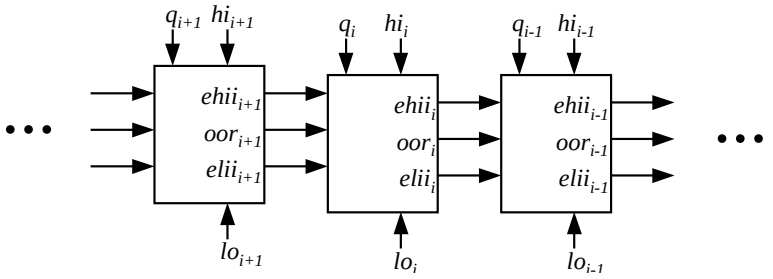

Figure 6: Iterative structure of revised range check circuit

The basic idea is this: once the query value is determined to be out of range, we no longer need the "equals *hi*" or "equals *lo*" signals. We can relax the constraints on the signals accordingly, as follows:

$$ehii_i \equiv \begin{cases} 0 & \text{if, up to and including bit } i,\ q \text{ is less than } hi \\ 1 & \text{if, up to and including bit } i,\ q \text{ is equal to } hi \\ \textit{undefined} & \text{if, up to and including bit } i,\ q \text{ is greater than } hi \end{cases}$$

$$elii_i \equiv \begin{cases} 0 & \text{if, up to and including bit } i,\ q \text{ is greater than } lo \\ 1 & \text{if, up to and including bit } i,\ q \text{ is equal to } lo \\ \textit{undefined} & \text{if, up to and including bit } i,\ q \text{ is less than } lo \end{cases}$$

$$oor_i \equiv \begin{cases} 0 & \text{if, up to and including bit } i,\ q \text{ is neither above } hi \text{ nor below } lo \\ 1 & \text{if, up to and including bit } i,\ q \text{ is either above } hi \text{ or below } lo \end{cases}$$

The inter-stage signal names are abbreviations as in Section 3, except *eh* and *el* have the string *ii* appended as a reminder that those signals are only valid **i**f the query is **i**n range. We use as initial conditions that $ehii_W$ and $elii_W$ are asserted, and oor_W is not.

As before, the final answer is determined simply by looking at the value of oor_0. This signal is asserted if and only if the query is out of range.

Since $ehii_i$ and $elii_i$ are undefined in some cases, we have more freedom in the implementation of this circuit than the previous one. The following set of logic expressions, for example, can be used:

$$ehii_i \equiv ehii_{i+1} \wedge (q_i \vee hi_i')$$

$$elii_i \equiv elii_{i+1} \wedge (q_i' \vee lo_i)$$

$$oor_i \equiv oor_{i+1} \vee (eh_{i+1} \wedge hi'_i \wedge q_i) \vee (el_{i+1} \wedge lo_i \wedge q'_i)$$

The expression defining $ehii_i$ differs from the Section 3 definition of eh_i by replacing the quantity $(q_i = hi_i)$ with the quantity $(q_i \vee hi_i)$. The value of these quantities differ only when $q_i = 1$ and $hi_i = 0$, and this difference is only relevant when $ehii_{i+1} = 1$. But in that case, the query will be out of range (as detected by oor_i), which means that $ehii_i$ is undefined. Thus a value of 1 under those conditions is acceptable. A similar analysis applies to the expression for $elii_i$.

A circuit for generating oor_i is shown in Figure 7; it is essentially the portion of the previous design (Figure 5) that computes oor_i. This requires 14 transistors, plus 6 for the inverters used to generate $ehii'_{i+1}$, $elii'_{i+1}$, and oor'_{i+1}. Figure 8 shows the logic required to generate $ehii_i$, which requires 6 transistors. Figure 9 shows the logic required to generate $elii_i$, which also requires 6 transistors. Adding the 12 transistors for the two SRAM storage cells (for hi_i and lo_i) brings the total to 44 transistors per bit, using this design.

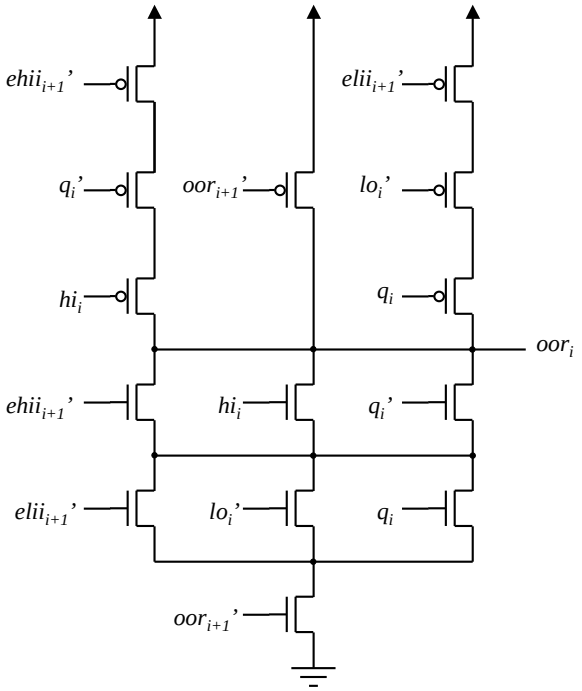

Figure 7: Sub-circuit for *out-of-range* signal

As in the previous design, the first stage circuitry can be simplified, given that the values of $ehii_W$, $elii_W$, and oor_W are fixed. The last stage can also be simplified, because the signals $ehii_0$ and $elii_0$ need not be generated.

5. Subfield Chaining

The aforementioned range match circuits are a good choice for a device that targets a particular application, where the number and width of range fields are known in advance. For situations where the manufacturer of a classification device does not know those parameters in advance, it would be nice to have some flexibility. One approach to this is to implement range matching via a set of smaller fields, chaining them together as needed. For example, if a device is built with subfields of

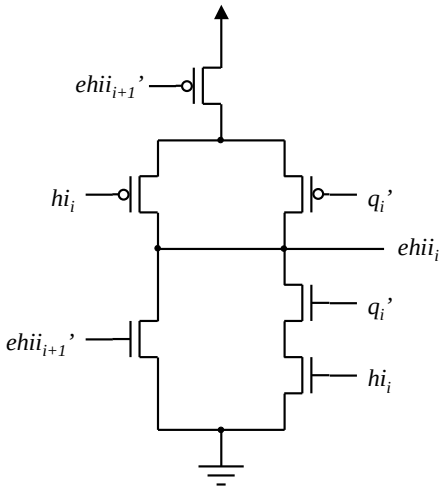

Figure 8: Sub-circuit for *equals-hi-if-in-range* signal

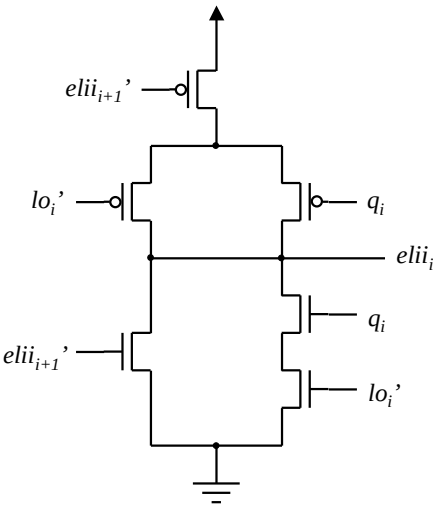

Figure 9: Sub-circuit for *equals-lo-if-in-range* signal

8 bits in width, then two such subfields can be chained together to perform a 16-bit range match, three can be chained to perform a 24-bit match, and so on.

This can be accomplished as follows: Consider a device with j subfields, each of width s and numbered $j-1$ through 0. Let the signal ch_i denote whether subfield i is chained to subfield $i-1$. Chaining subfield i involves an interaction between the stage for the least significant bit of subfield i and the stage for the most significant bit of subfield $i-1$. To accomplish this, each stage $si-1$ (for each integer i, where $1 \leq i \leq j$) uses signal ch_i to determine whether to use the inter-stage signals from stage si as inputs or not; if not, then the normal initial conditions are used (e.g. for the design in Section 4, use 1 in place of $ehii_{si}$ and $elii_{si}$, and 0 in place of oor_{si}.) An example of logic for generating oor_{si} for stage $si-1$ is shown in Figure 10. Figure 11 and figure 12 show the logic for $ehii_{si}$ and $elii_{si}$ respectively.

Whether all of the range fields in a word match can be determined by examining the *oor* signals for all subfields (i.e. oor_{si} for all integers i where $0 \leq i < j$), regardless of how the subfields are chained. A match is indicated when none of those signals are asserted. This works because, in a multi-subfield match, the first s bits must match, the first $2s$ bits, and so on, if the entire quantity matches.

There is a tradeoff between subfield width and versatility of the range matching device. A smaller subfield requires more overhead (for the additional "chaining enable" signals and logic), but results in finer granularity with respect to allocation of bits for range match fields; this can result in more efficient use of the available bits.

In the extreme case where subfield width equals one, range match fields can be created of any arbitrary width without wasting bits. Also in that case, range match bits can perform the same type of matching as standard TCAM bits, thus providing even greater flexibilty. This style of matching is accomplished by using fields of width 1 and using lower and upper bounds of (0, 0) to represent 0, (1, 1) to represent 1, and (0, 1) to represent "don't care."

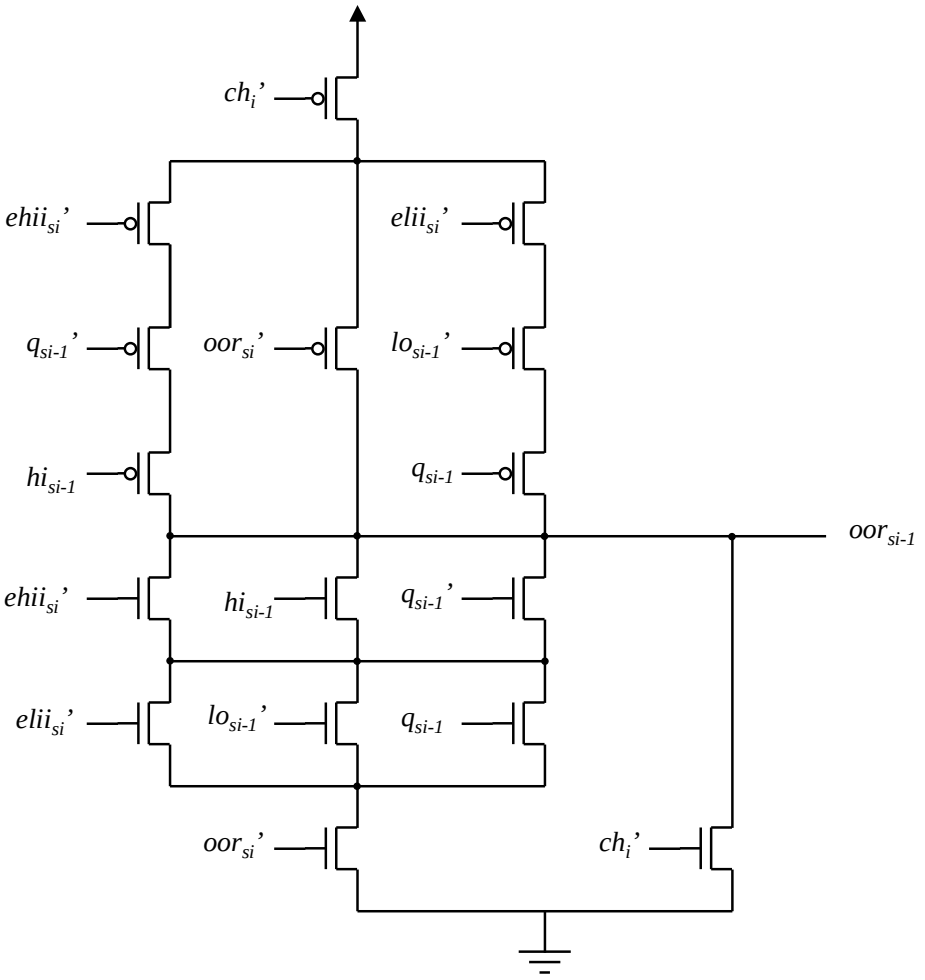

Figure 10: Sub-circuit for *out-of-range* signal for input to stage $si - 1$

6. Conclusion

We have described three CMOS implementations of a range check circuit, the most efficient of which requires 44 transitors per bit. For a IPv4 application, this means a 46% increase in transistor count per word, but for the filter sets studies in [1], it also means using a sixth to half as many words; in those cases, overall transistor count required is only 24% to 78% of the standard TCAM design, and power dissipation is reduced similarly. Also, the range check support greatly simplifies creation and maintenance of the TCAM contents, since there is now a one-to-one correspondence between filters and TCAM entries. In addition, we describe a means of chaining small range match subfields together; thus a range matching device can be configured for various numbers and sizes of range match fields, by chaining subfields as needed. In the most extreme case where subfields are one bit wide, the range match portion of a device can also perform standard TCAM-style matching.

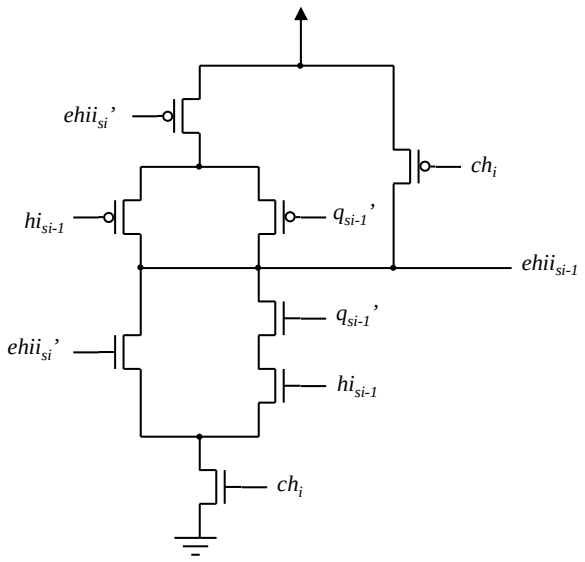

Figure 11: Sub-circuit for *equals-hi-if-in-range* signal for input to stage $si - 1$

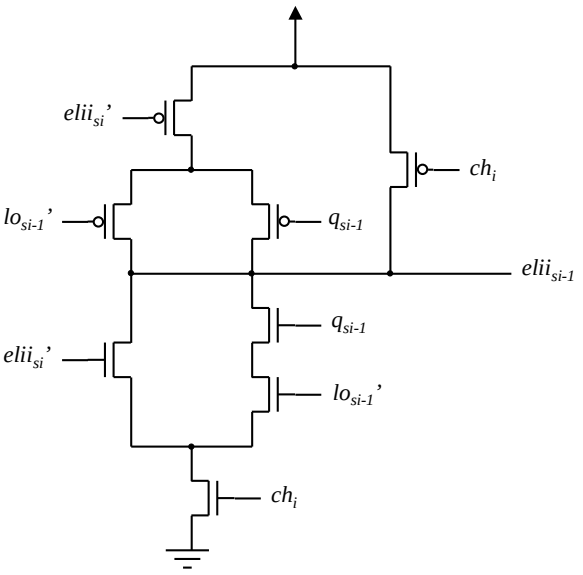

Figure 12: Sub-circuit for *equals-lo-if-in-range* signal for input to stage $si - 1$

References

[1] E. Spitznagel, D. Taylor, J. Turner, "Packet Classification Using Extended TCAMs," ICNP 2003.

[2] E. Spitznagel, "High Performance Packet Classification," Washington University Department of Computer Science and Engineering, Technical Report WUCSE-2004-14, March 2004.

[3] Someone, "Some TCAM references maybe," wherever they are.

[4] D. Taylor, "Survey & Taxonomy of Packet Classification Techniques," *to be submitted to ACM Computing Surveys.*

[5] J. van Lunteren, T. Engbersen, "Fast and Scalable Packet Classification," IEEE Journal on Selected Areas in Communication 21, 4 (May), 560-571.