

Informal Specification of Bitwalker

Andreas Carben, Jens Gerlach, Kim Völlinger

October 29, 2013

Contents

1	Introduction	2
2	Primary Functions of Bitwalker	3
2.1	The Function <code>Bitwalker_Peek</code>	3
2.2	The Function <code>Bitwalker_Poke</code>	4
2.3	Interaction of <code>Bitwalker_Peek</code> and <code>Bitwalker_Poke</code>	5
3	Secondary Functions of Bitwalker	6
3.1	The Data Structure <code>T_Bitwalker_Incremental_Locals</code>	6
3.2	The Function <code>Bitwalker_IncrementalWalker_Init</code>	6
3.3	The Function <code>Bitwalker_IncrementalWalker_Peek_Next</code>	6
3.4	The Function <code>Bitwalker_IncrementalWalker_Peek_Finish</code>	6
3.5	The Function <code>Bitwalker_IncrementalWalker_Poke_Next</code>	6
3.6	The Function <code>Bitwalker_IncrementalWalker_Poke_Finish</code>	6

List of Corrections

Fatal: to be done in later sprint	6
---	---

1 Introduction

We introduce some auxiliary concepts and formulate general assumptions:

- A *bit stream* is an array containing elements of type `uint8_t`.
A bit stream of length n contains $8n$ bits.
- A bit stream is *valid* if the array is valid.
- A bit stream can be indexed both by its array indices and its *bit indices*.

Figure 1 shows the difference between array indices and bit indices in a bit stream. The two bit indices, 0 and 14, mark bit positions in the first and second array element, respectively.

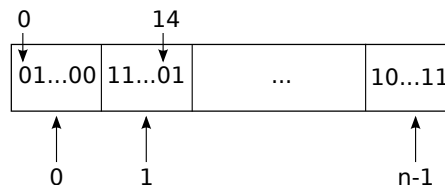


Figure 1: Array indices and bit indices in a bit stream

- A *bit sequence* is a consecutive sequence of bits within a bit stream as represented in Figure 2.

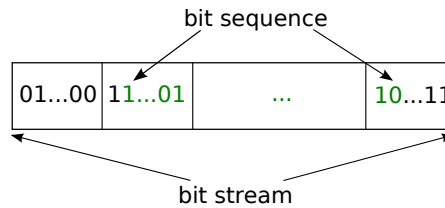


Figure 2: A bit sequence within a bit stream

A bit sequence is given by the position of its first bit (a bit index in the bit stream) and its *length*, that is, the number of bits it contains.

- A bit sequence of length l that starts at bit index p is *valid* with respect to a bit stream of length n if the following conditions are satisfied

$$\begin{aligned} 0 &\leq p \leq 8n \\ 0 &\leq p + l \leq 8n \end{aligned}$$

- We assume that the C-types `unsigned int` and `int` have a width of 32 bits.

2 Primary Functions of Bitwalker

The core functionality of the bitwalker is expressed by the two functions `Bitwalker_Peek` and `Bitwalker_Poke`.

2.1 The Function `Bitwalker_Peek`

The function `Bitwalker_Peek` reads a bit sequence from a bit stream and converts it to an integer.

Its function signature reads as follows:

```
uint64_t Bitwalker_Peek(unsigned int Startposition,  
                        unsigned int Length,  
                        uint8_t Bitstream[],  
                        unsigned int BitstreamSizeInBytes);
```

Arguments

- `Startposition` is the bit index in the bit stream where the bit sequence starts.
- `Length` is the length of the bit sequence.
- `Bitstream` is the array which provides the bit stream.
- `BitstreamSizeInBytes` is the length of the array containing the bit stream.

Preconditions

The following preconditions shall hold for the function arguments:

- `Bitstream` is a valid array of length `BitstreamSizeInBytes`
- $\text{Length} \leq 64$ and
- $\text{Startposition} + \text{Length} \leq \text{UINT_MAX}$.

Note that additional constraints are implicitly expressed by the use of *unsigned* integer types.

Description

The function `Bitwalker_Peek` reads a bit sequence from a bit stream and converts it to a 64-bit unsigned integer.

The left most bit of the bit sequence is interpreted as the most significant bit. Thus, for a bit sequence $(b_0, b_1, \dots, b_{n-1})$ the function returns the sum

$$b_0 \cdot 2^{n-1} + b_1 \cdot 2^{n-2} + \dots + b_{n-1} \cdot 2^0 = \sum_{i=0}^{n-1} b_i \cdot 2^{(n-1)-i} \quad (1)$$

If the bit sequence is not valid, then the function returns 0. This increases the robustness of the function.

2.2 The Function `Bitwalker_Poke`

The function `Bitwalker_Poke` converts an integer to a bit sequence and writes it into a bit stream. Its function signature reads as follows:

```
int Bitwalker_Poke(unsigned int Startposition,
                  unsigned int Length,
                  uint8_t Bitstream[],
                  unsigned int BitstreamSizeInBytes,
                  uint64_t Value);
```

Arguments

- `Startposition` is the bit index in the bit stream where the bit sequence starts.
- `Length` is the length of the bit sequence.
- `Bitstream` is the array which provides the bit stream.
- `BitstreamSizeInBytes` is the length of the array containing the bit stream.
- `Value` is the integer which shall be converted into a bit sequence.

Preconditions

The following preconditions shall hold for the function arguments:

- `Bitstream` is a valid array of length `BitstreamSizeInBytes`
- `Startposition + Length ≤ UINT_MAX`.

Note that additional constraints are implicitly expressed by the use of *unsigned* integer types.

Description

The function `Bitwalker_Poke` converts a 64-bit unsigned integer to a bit sequence and writes it into a bit stream.

For $0 \leq x$ exists a shortest sequence of 0 and 1 $(b_0, b_1, \dots, b_{n-1})$ such that

$$\sum_{i=0}^{n-1} b_i \cdot 2^{(n-1)-i} = x. \quad (2)$$

The function `Bitwalker_Poke` tries to store the sequence $(b_0, b_1, \dots, b_{n-1})$ in the bit sequence of `Length` bits that starts at bit index `Startposition`.

The return value of `Bitwalker_Poke` depends on the following three cases:

- If the bit sequence is valid, then there are two cases:
 - If $\text{Length} \geq n$, then the sequence $(\overbrace{0, \dots, 0}^{\text{Length}-n}, b_0, b_1, \dots, b_{n-1})$ is stored in the bit stream starting at `Startposition`. The return value of `Bitwalker_Poke` is 0.
 - If $\text{Length} < n$, then the sequence $(b_0, b_1, \dots, b_{n-1})$ cannot be stored and `Bitwalker_Poke` returns -2.
- If the bit sequence is not valid, then `Bitwalker_Poke` returns -1.

2.3 Interaction of `Bitwalker_Peek` and `Bitwalker_Poke`

The functions `Bitwalker_Peek` and `Bitwalker_Poke` are inverse to each other.

3 Secondary Functions of Bitwalker

FiXme Fatal: to be done in later sprint

3.1 The Data Structure `T_Bitwalker_Incremental_Locals`

3.2 The Function `Bitwalker_IncrementalWalker_Init`

3.3 The Function `Bitwalker_IncrementalWalker_Peek_Next`

3.4 The Function `Bitwalker_IncrementalWalker_Peek_Finish`

3.5 The Function `Bitwalker_IncrementalWalker_Poke_Next`

3.6 The Function `Bitwalker_IncrementalWalker_Poke_Finish`