Using Program Specialization to Speed SystemC Fixed-Point Simulation

Stephen A. Edwards

Department of Computer Science,
Columbia University

www.cs.columbia.edu/~sedwards

sedwards@cs.columbia.edu

Signal Processing Algorithms

Very simple, mathematically speaking:

$$y_t = \sum_{i=0}^{n-1} a_i x_{t-i}$$

n-tap FIR filter

$$S_n(u) = \frac{C_u}{2} \sum_{x=0}^{n-1} f(x) \cos \frac{(2x+1)\pi u}{16}$$

n-point IDCT

$$f_j = \sum_{k=0}^{n-1} x_k e^{-\frac{2\pi i}{n} jk}$$

n-point DFT

FIR in SystemC

```
#include "systemc.h"
#define N 25

double fix_fir(double _in[])
{
  sc_fxtype_params param(32, 16, SC_RND, SC_SAT);
  sc_fxtype_context con(param);

  sc_fix in[N], c[N], t[N], y;

  int i;                         /* Init coefficients */
  double ct = 0.9987966;
  for (i = 0 ; i<N ; i++) {
    in[i] = _in[i]; c[i] = ct; ct /= 2;
  }

  for (i = 0 ; i < N ; i++) {    /* Dot product */
    t[i] = c[i] * in[i];         /* Fixed-point multiplication */
    y += t[i];                   /* Fixed-point addition */
  }

  return y;                      /* Type conversion */
}
```

What I Did

SystemC fixed-point code can be 1000$\times$ slower than floating-point. What can partial evaluation recover?

Selected Consel et al.'s Tempo + Prespec.

SystemC a C++ library; manually (stupidly) rewrote it in C.

Improvement from Tempo: 1.8$\times$.

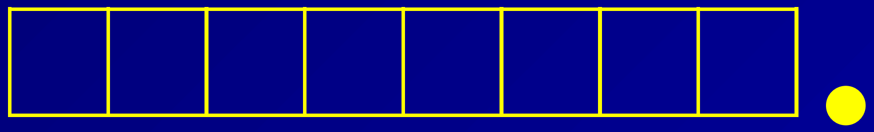
Rewrote library for specialization.

Improvement from Tempo: 3–6$\times$

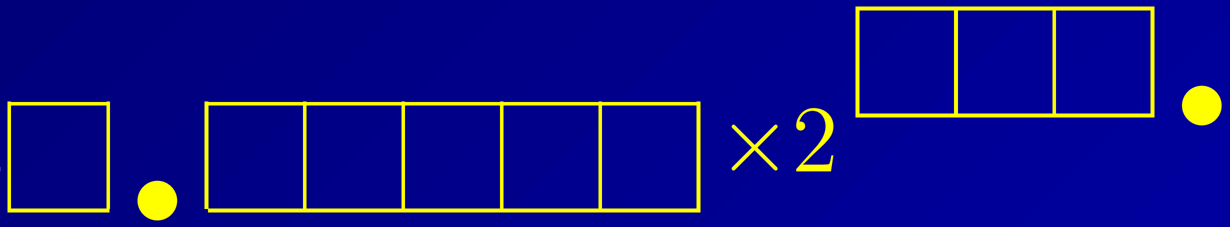
3–6$\times$ slower than floating-point, comparable to Meyr et al.'s Fridge (a custom code generator)

Integers, Floating-point, Fixed-point

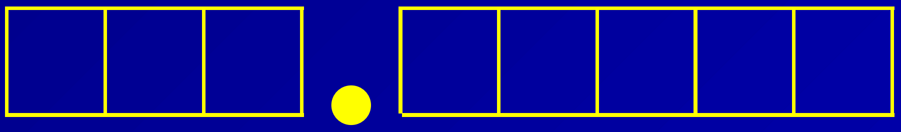
Integer



Floating-point



Fixed-point



SystemC's Fixed-Point Types

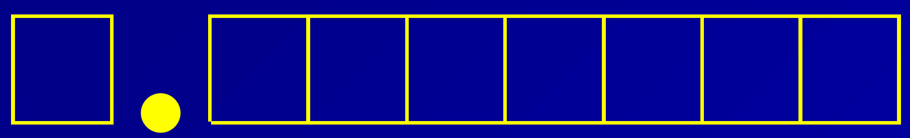
```
sc_fixed<8, 1, SC_RND, SC_SAT> fpn;
```

8 is the total number of bits in the type

1 is the number of bits to the left of the decimal point

SC_RND defines rounding behavior

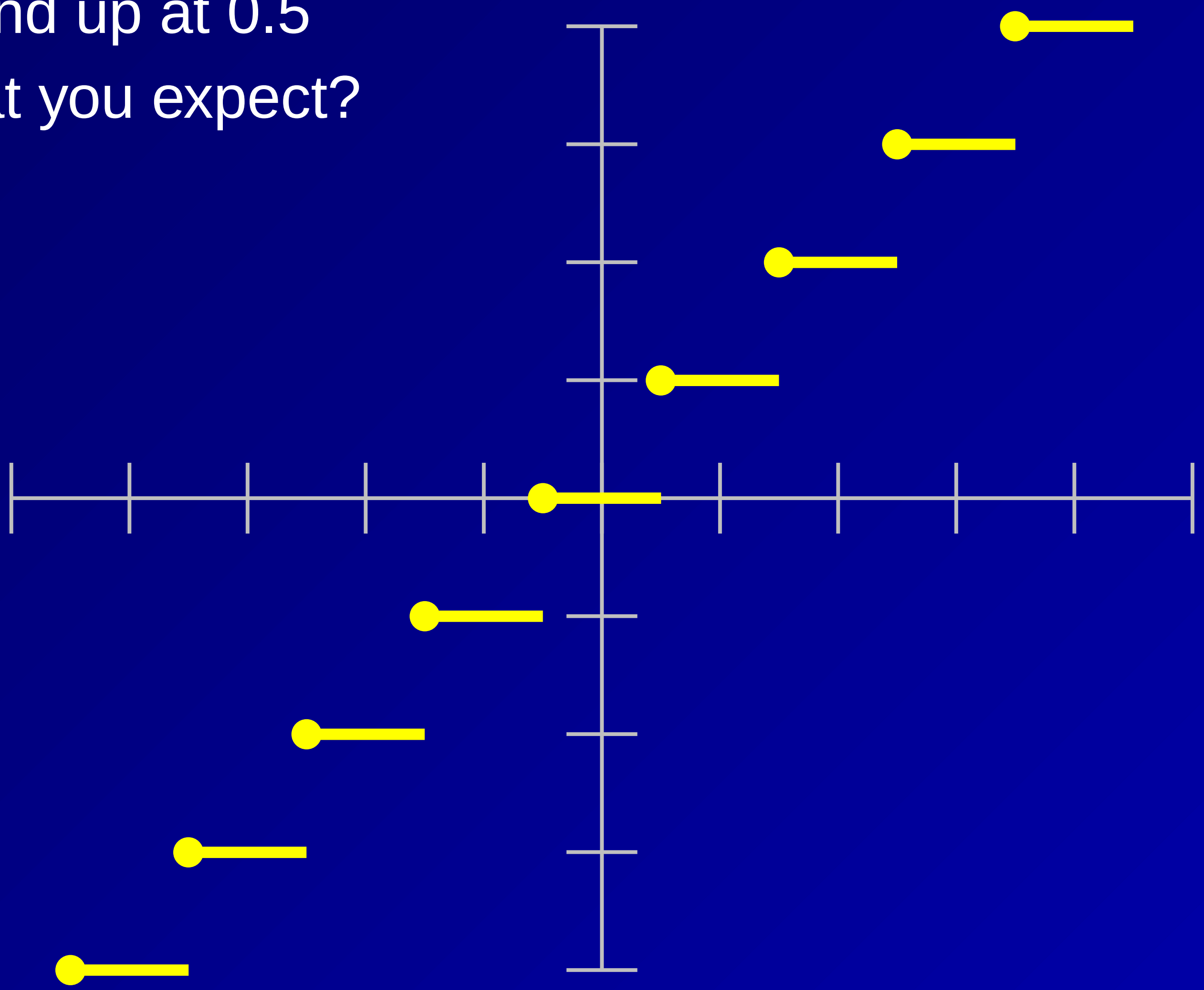
SC_SAT defines saturation behavior



Rounding Modes: SC_RND

Round up at 0.5

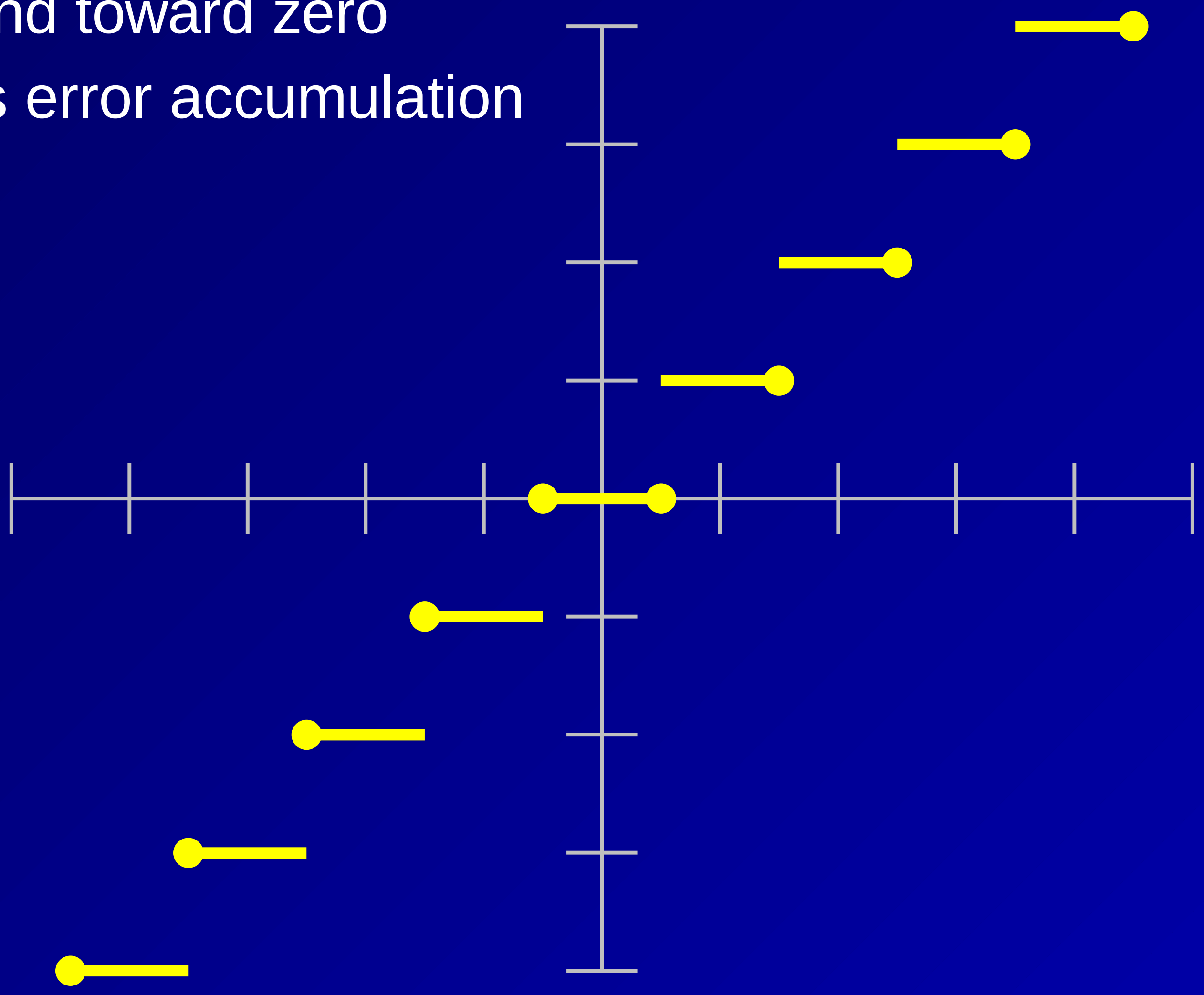
What you expect?



Rounding Modes: SC_RND_ZERO

Round toward zero

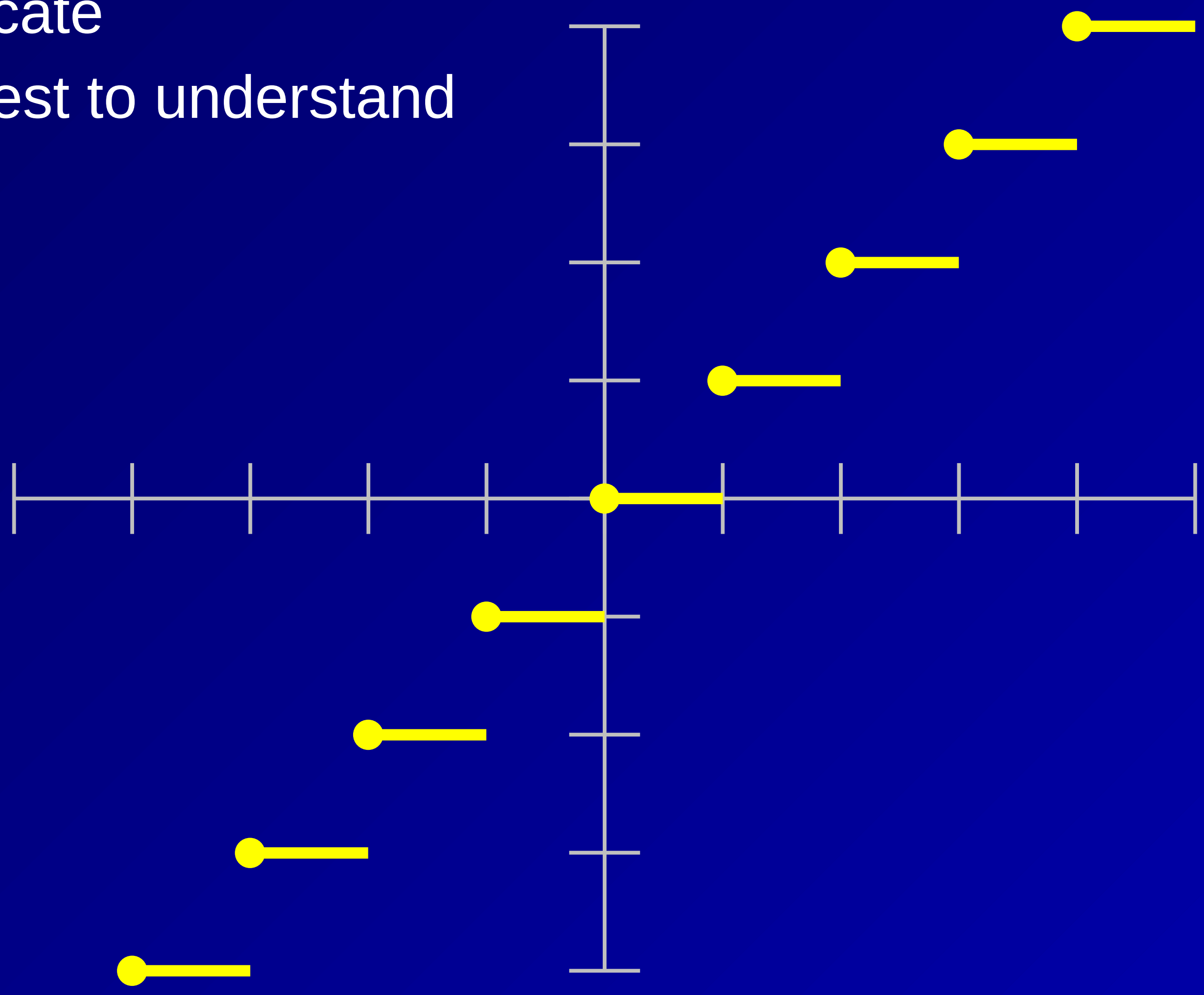
Less error accumulation



Rounding Modes: SC_TRN

Truncate

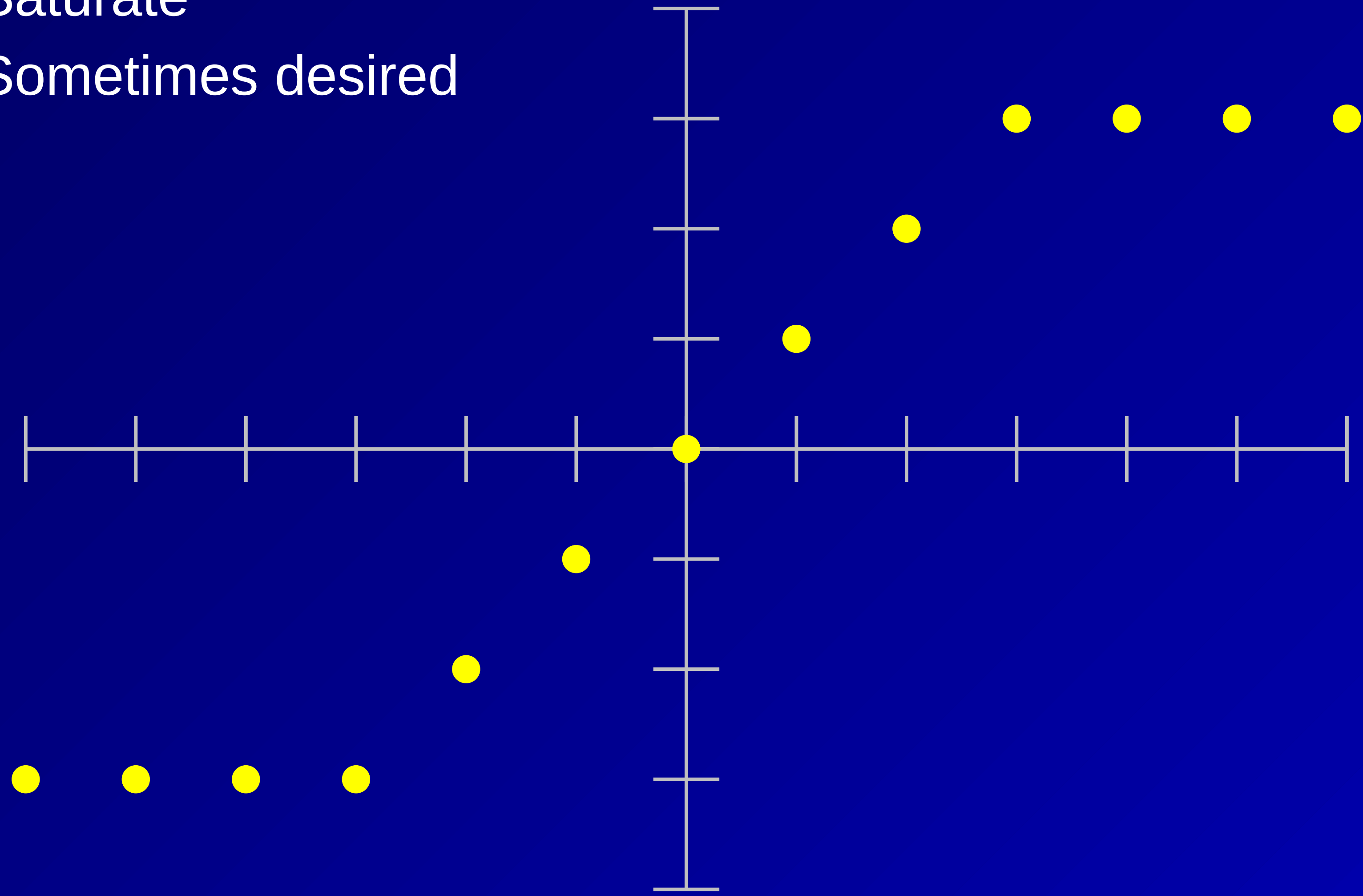
Easiest to understand



Overflow Modes: SC_SAT

Saturate

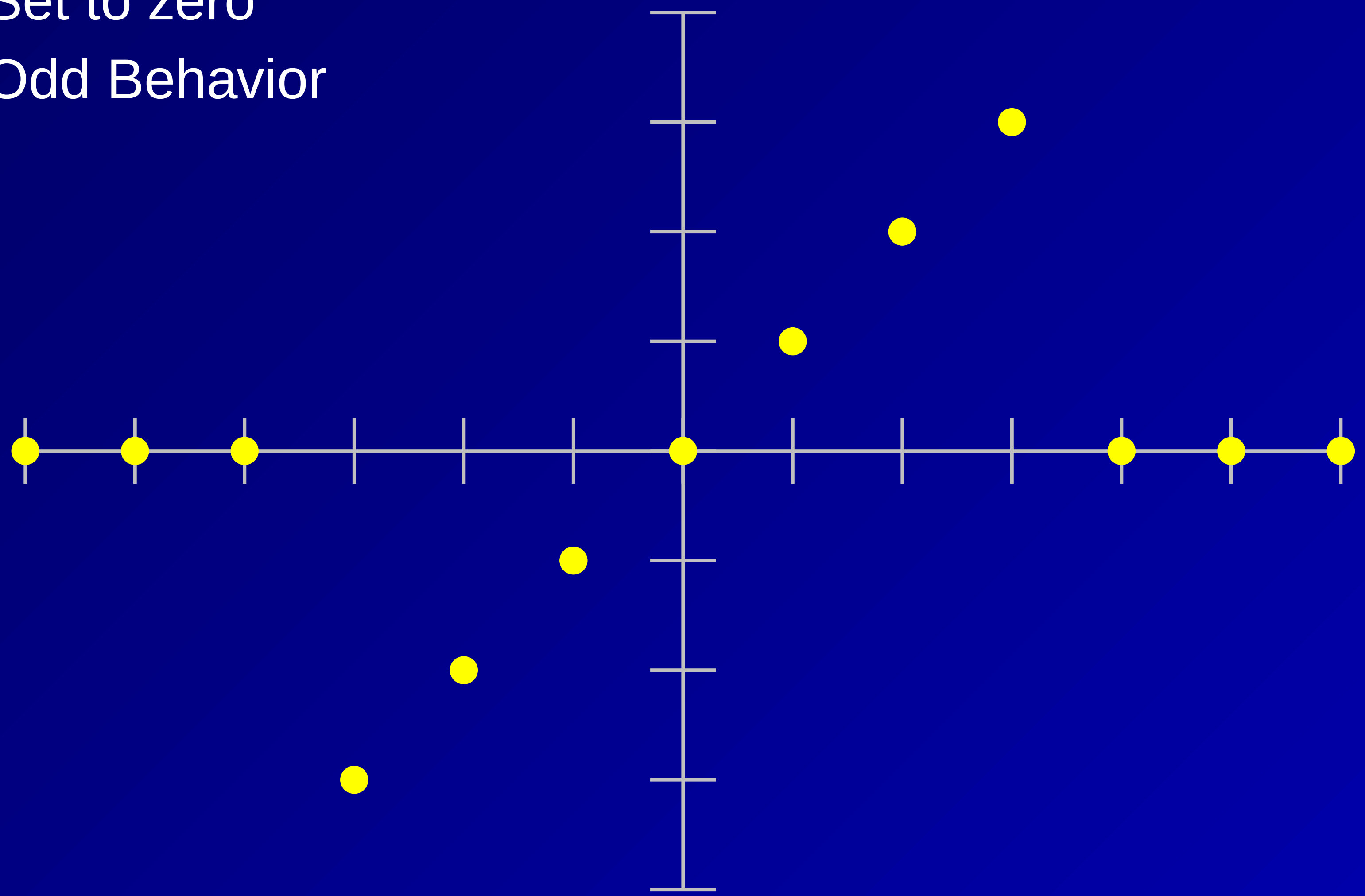
Sometimes desired



Overflow Modes: SC_SAT_ZERO

Set to zero

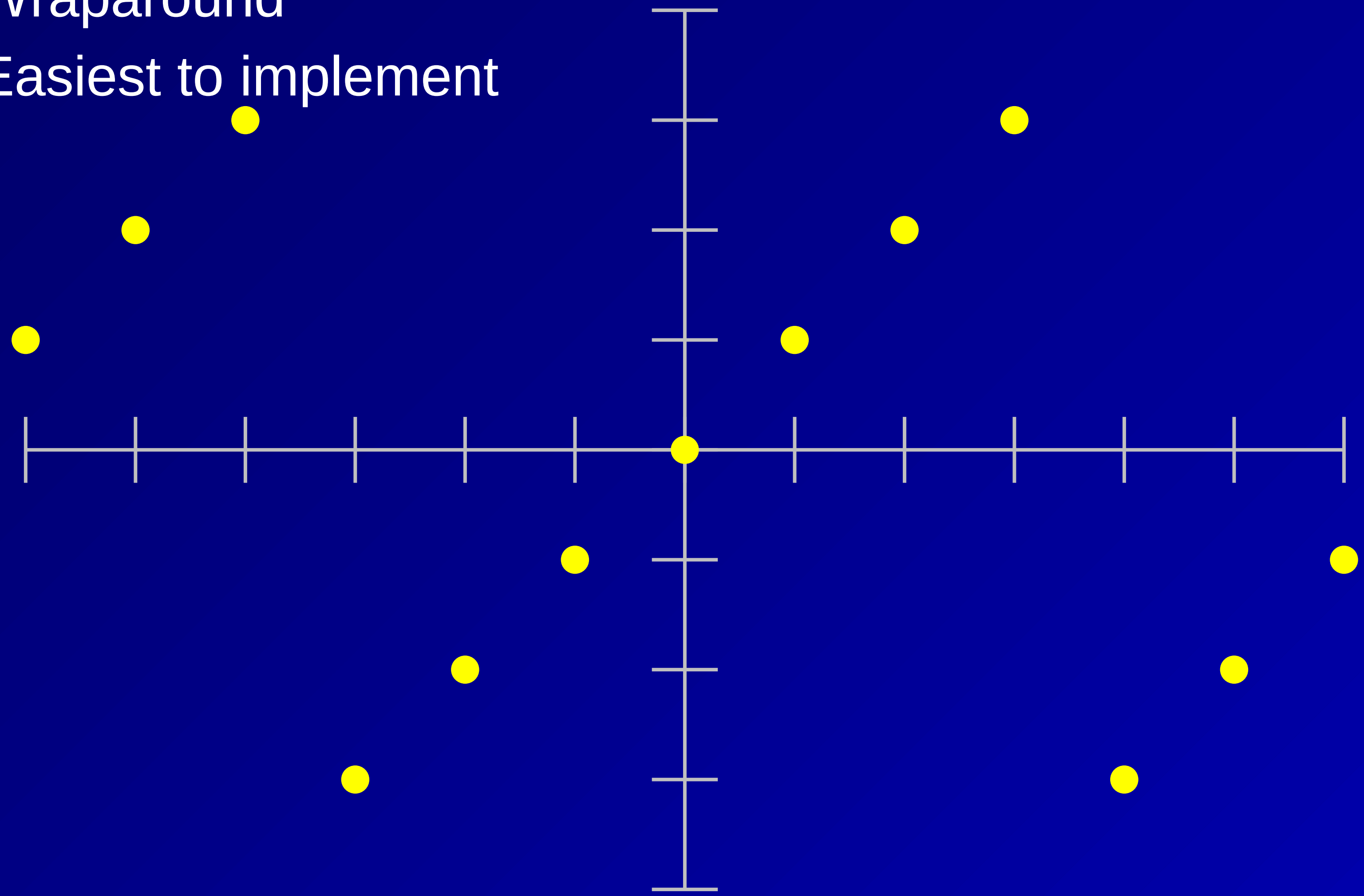
Odd Behavior



Overflow Modes: SC_WRAP

Wraparound

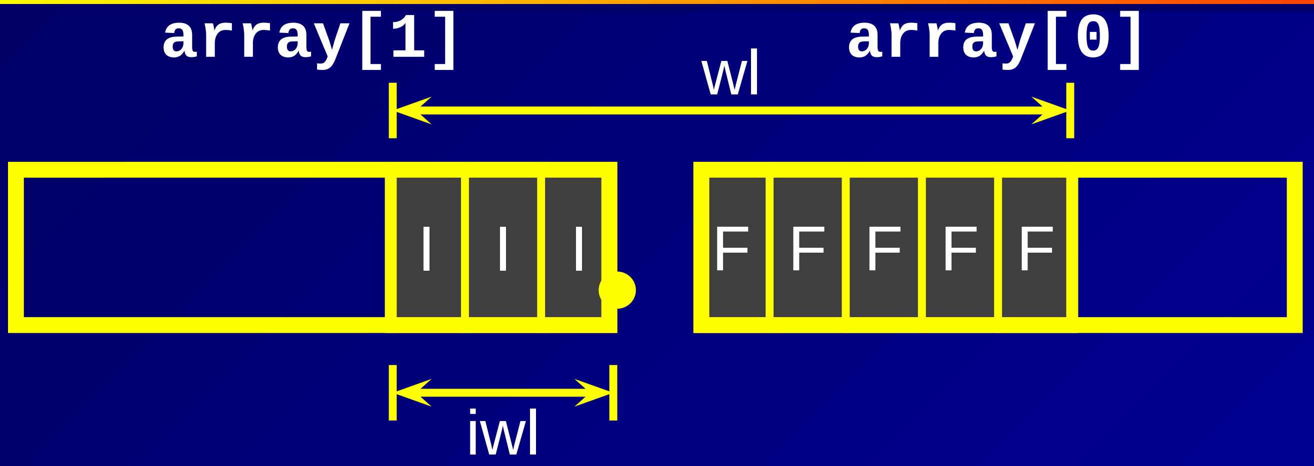
Easiest to implement



Experimental Results

	Times			Speedup			vs. recoded		
	FIR	IDCT	FFT	vs. SystemC			in C		
SystemC	26000	41000	110000	1	1	1			
Recoded in C	6300	6100	34000	4.2	6.6	3.4	1	1	1
Specialized	3700	3400	18000	7.1	12	6.1	1.7	1.8	1.8
Doubles	290	40	420	92	1000	270	22	150	80
Floats	260	40	380	100	1000	300	25	150	88

The SystemC Representation



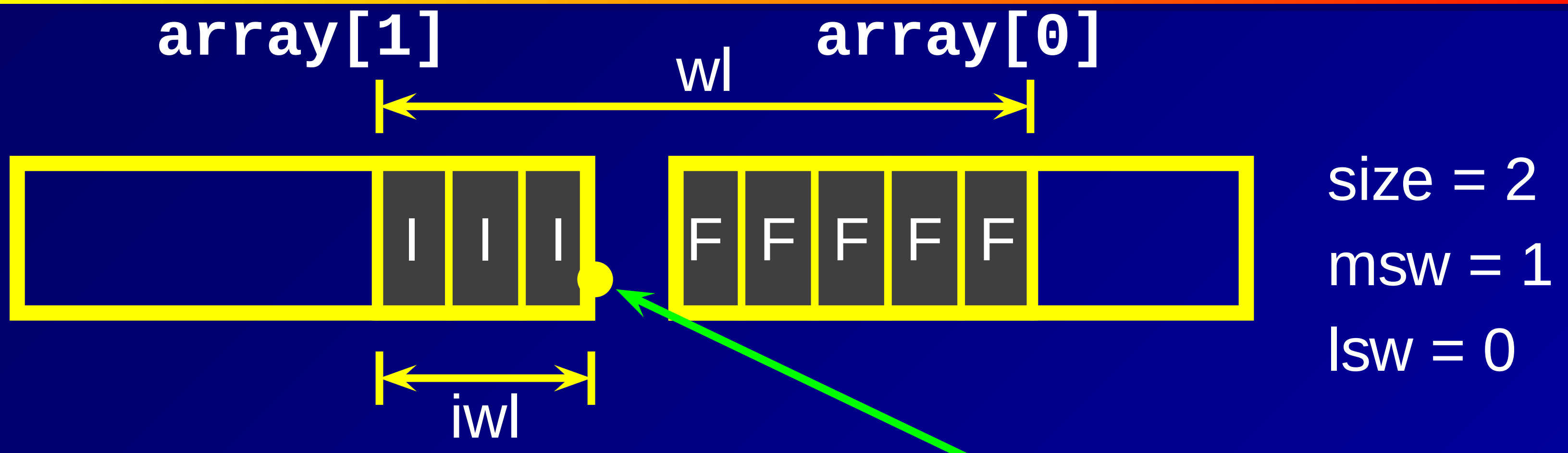
size = 2

msw = 1

lsw = 0

```
typedef unsigned long word;
typedef struct fixed {
   word *array; /* mantissa array */
   int size;    /* words in the array */
   int q_mode;  /* Quantization mode */
   int o_mode;  /* Overflow mode */
   int state;   /* Current state */
   int wp;      /* units word index */
   int sign;    /* 1 or -1 */
   int msw;     /* most significant word */
   int lsw;     /* least significant word */
   int wl;      /* word length */
   int iwl;     /* integer word length */
} fixed_fix;
```

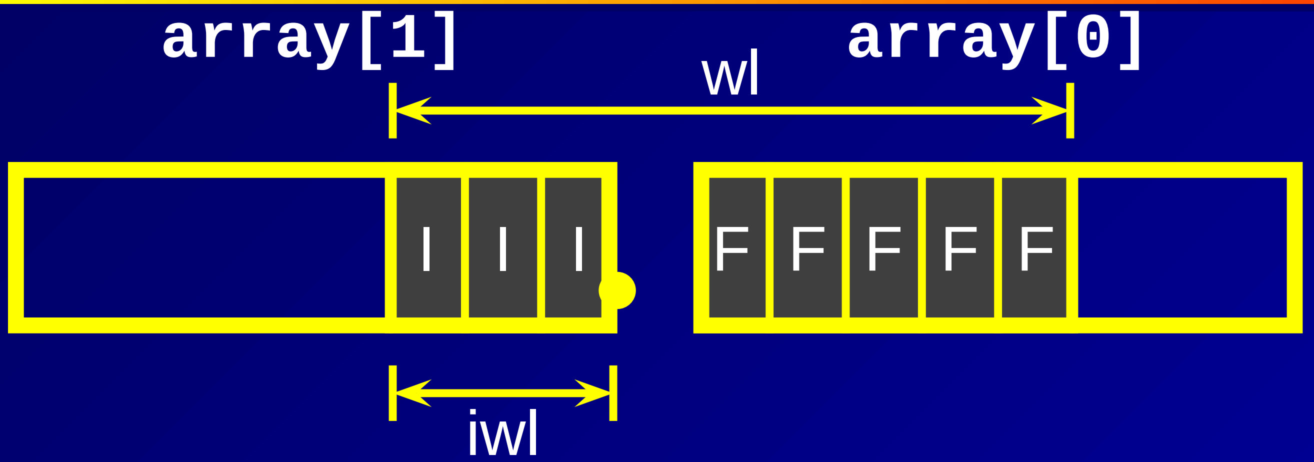
The SystemC Representation




```
typedef unsigned long word;
typedef struct fixed {
   word *array; /* mantissa array */
   int size;    /* words in the array */
   int q_mode;  /* Quantization mode */
   int o_mode;  /* Overflow mode */
   int state;   /* Current state */
   int wp;      /* units word index */
   int sign;    /* 1 or -1 */
   int msw;     /* most significant word */
   int lsw;     /* least significant word */
   int wl;      /* word length */
   int iwl;     /* integer word length */
} fixed_fix;
```

Decimal point location avoids shifts but forces operations to manipulate two words.

The SystemC Representation



size = 2

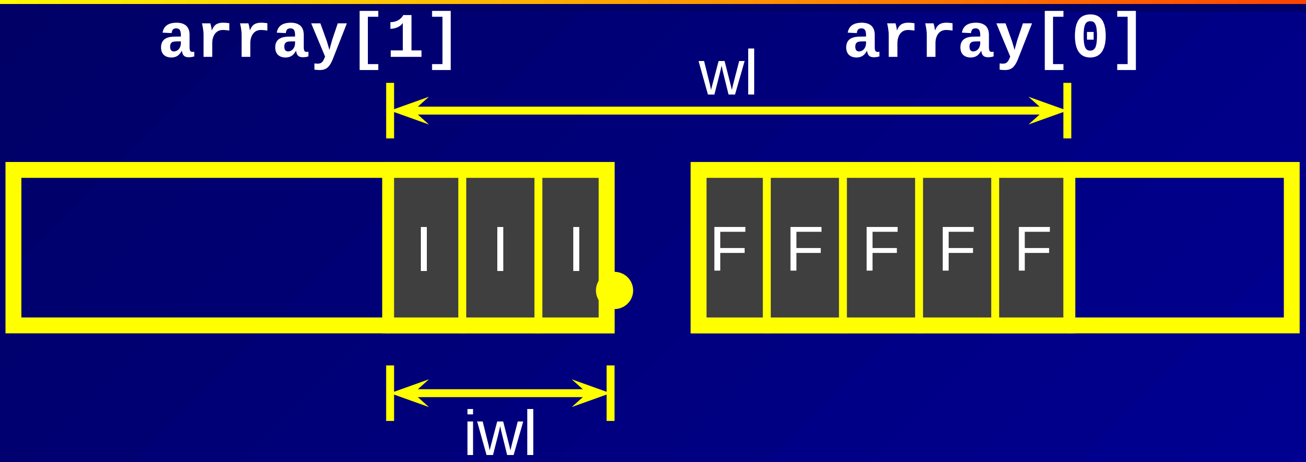
msw = 1

lsw = 0

```
typedef unsigned long word;
typedef struct fixed {
   word *array; /* mantissa array */
   int size;    /* words in the array */
   int q_mode;  /* Quantization mode */
   int o_mode;  /* Overflow mode */
   int state;   /* Current state */
   int wp;      /* units word index */
   int sign;    /* 1 or -1 */
   int msw;     /* most significant word */
   int lsw;     /* least significant word */
   int wl;      /* word length */
   int iwl;     /* integer word length */
} fixed_fix;
```

msw and lsw indicate which words are in use. Constant for almost all numbers, yet cannot be specialized.

The SystemC Representation



size = 2

msw = 1

lsw = 0

```
typedef unsigned long word;
typedef struct fixed {
   word *array; /* mantissa array */
   int size;    /* words in the array */
   int q_mode;  /* Quantization mode */
   int o_mode;  /* Overflow mode */
   int state;   /* Current state */
   int wp;      /* units word index */
   int sign;    /* 1 or -1 */
   int msw;     /* most significant word */
   int lsw;     /* least significant word */
   int wl;      /* word length */
   int iwl;     /* integer word length */
} fixed_fix;
```

Maintaining explicit sign bit as costly as mantissa for small numbers

Other Challenges

In C, **`int * int = int`** (higher-order bits truncated)

32 $\times$ 32-bit multiplication: four mults plus masks & shifts.

SystemC libraries use a trick to convert to/from **`double`**:

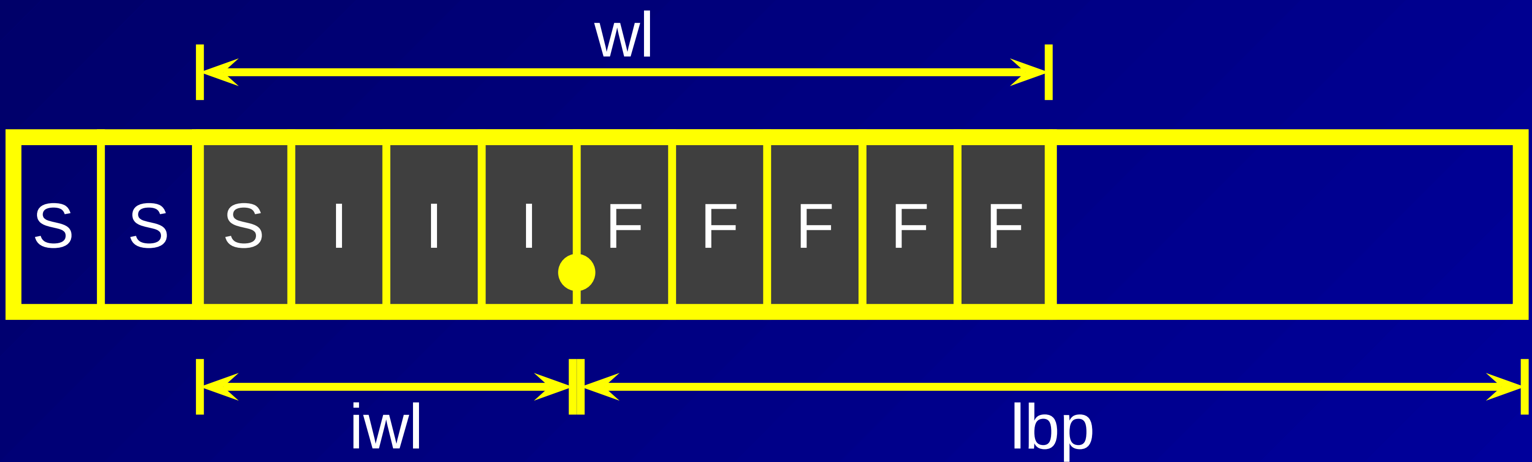
```
union ieee_double {
  double d;
  struct {
    unsigned negative : 1;
    unsigned exponent : 11;
    unsigned mantissa0 : 20;
    unsigned mantissa1 : 32;
  } s;
};
```

(Write to d, read from fields of s & vice versa)

Conclusion: Tempo cannot change representations, so can't do much with SystemC code.

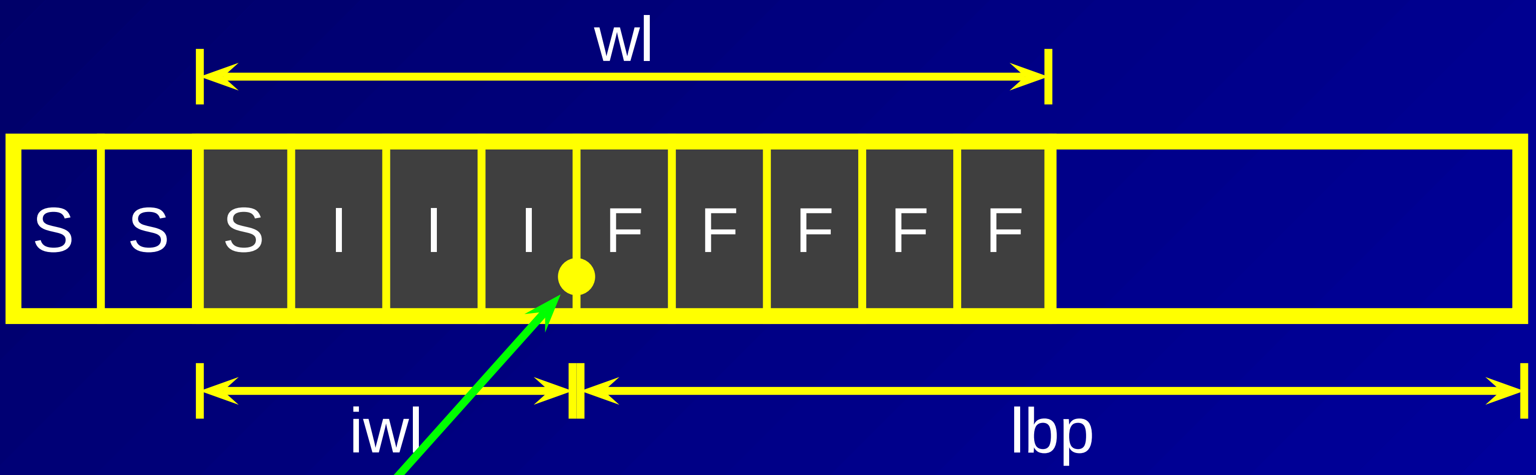
Solution: Try rewriting it for specialization.

The FRIDGE Representation




```
typedef struct fp {
   int val;      /* 32-bit value */
   int wl;       /* Word length (bits) */
   int iwl;      /* Integer word length (bits) */
   int lbp;      /* Location of binary point (bits) */
   int overflow;
   int rounding;
} fixed;
```

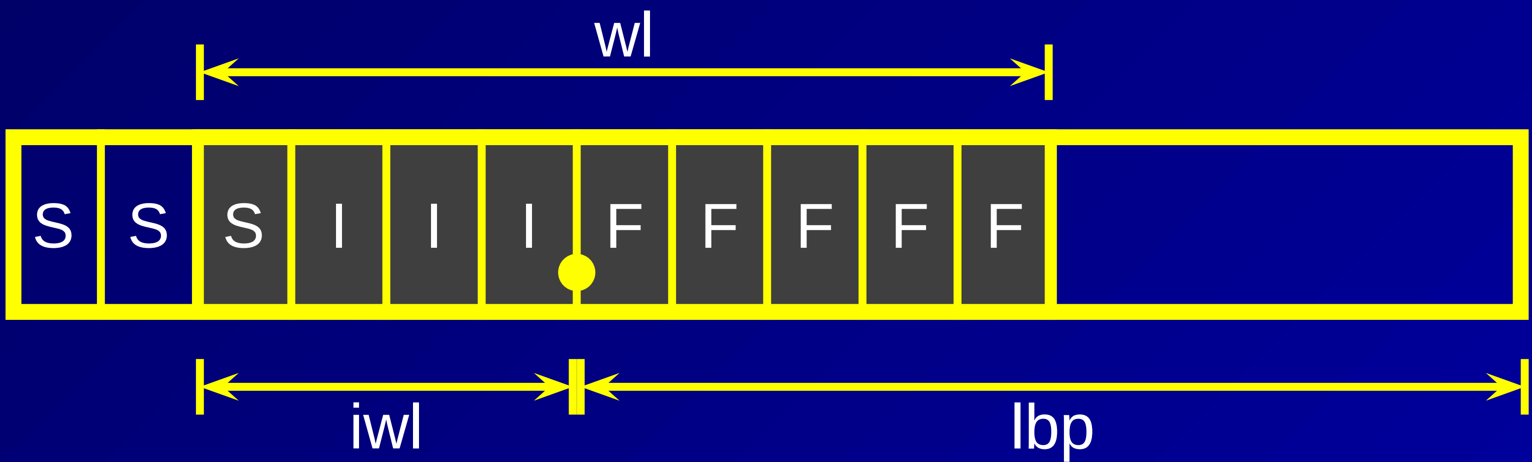
The FRIDGE Representation




```
typedef struct fp {
   int val;      /* 32-bit value */
   int wl;       /* Word length (bits) */
   int iwl;      /* Integer word length (bits) */
   int lbp;      /* Location of binary point (bits) */
   int overflow;
   int rounding;
} fixed;
```

Decimal point within a word requires shifting but permits single-word operations.

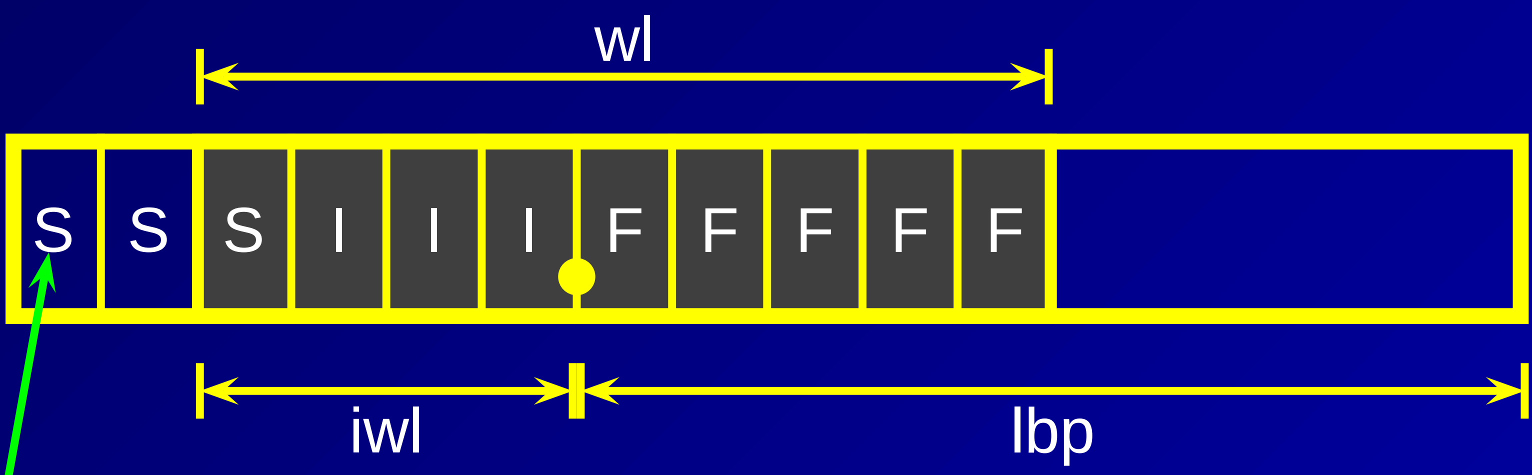
The FRIDGE Representation




```
typedef struct fp {
    int val;       /* 32-bit value */
    int wl;        /* Word length (bits) */
    int iwl;       /* Integer word length (bits) */
    int lbp;       /* Location of binary point (bits) */
    int overflow;
    int rounding;
} fixed;
```

Only val is dynamic; everything else can be specialized.

The FRIDGE Representation




```
typedef struct fp {
    int val;      /* 32-bit value */
    int wl;       /* Word length (bits) */
    int iwl;      /* Integer word length (bits) */
    int lbp;      /* Location of binary point (bits) */
    int overflow;
    int rounding;
} fixed;
```

Two's complement representation avoids additional sign fi eld.

Leads To Simple Code

```
void mult(fixed *r, fixed *a, fixed *b) {
  int av, bv, shift;
  av = a->val >> (a->lbp - (a->wl - a->iwl));
  bv = b->val >> (b->lbp - (b->wl - b->iwl));
  shift = (a->wl - a->iwl) +
          (b->wl - b->iwl) - r->lbp;
  r->val = av * bv;
  if (shift > 0) r->val >>= shift;
  else if (shift < 0) r->val <<= -shift;
  fix_quantize(r);
  fix_overflow(r);
}
```

Quantize()

```
void quantize(fixed *r) {
  int shift, delta, mask;
  switch (r->rounding) {
  case ROUND:
    delta = 1 << (r->lbp - (r->wl - r->iwl)) - 1;
    shift = r->lbp -  (r->wl - r->iwl);
    r->val = (r->val + delta) >> shift) << shift;
    break;
  case TRUNCATE:
    mask = 1 << (r->lbp - (r->wl - r->iwl)) - 1;
    r->val &= ~mask;
    break;
  }
}
```

After Specialization

```
void mult(fixed *r, fixed *a, fixed *b) {
  int av, bv;
  av = a->val >> 4;
  bv = b->val >> 4;
  r->val = av * bv;
  r->val >>= 8;
  r->val &= 0xffffff0;  /* From quantize() */
  if (r->val > 0x7fff0)  /* From overflow() */
    r->val = 0x7fff0;
  else if (r->val < -0x80011)
    r->val = -0x80011;
}
```

(wl=16, iwl=4, lbp=16, quant=TRUNC, overflow=SAT)

Experimental Results

	Times FIR	Times IDCT	Times FFT	Speedup vs. SystemC			vs. for specialization		
SystemC	26000	41000	110000	1	1	1			
Rewritten	2300	1500	8900	11	26	13	1	1	1
Library specialized	1000	570	3400	25	72	33	2.2	2.7	2.6
Program specialized	720	250	1300	36	160	86	3.2	6.1	6.8
Double precision fbats	290	40	420	92	1000	270	8	39	21
Single precision fbats	260	40	380	100	1000	300	9	39	23

Conclusions

- Specializing “mechanical” C translation gave only a 1.8$\times$ speedup, still 22–150$\times$ slower than **`double`**s
- Problem was poor choice of number representation
- Rewriting for specialization: 2.2–2.7$\times$ speedup
- Specializing program with libraries: 3.2–6.8$\times$
- Final result within a factor of 2.8–6.4$\times$ of **`double`**s
- Not ready for prime time