Holy City Audio Forum

Former Home of SpinCAD Designer

Optimizing Spin ASM code generation

Optimizing Spin ASM code generation (#p1673)

by **Digital Larry** » Thu Jun 26, 2014 5:48 am

Wonderful as it is, SpinCAD Designer generates pretty inefficient Spin ASM code. The main reason for this is that every output connection reserves one of the FV-1's 32 internal general purpose registers. I hadn't worried about it too much, but in the past few days I created a patch with so many blocks that I actually did run out of registers before anything else.

Let's look at an example.

```
;------ Smoother
RDAX 37,1.0
RDFX 40,0.0700
WRAX 40,0.0
;------ Scale/Offset
RDAX 40,1.0
SOF 0.2700,0.7300
WRAX 41,0.0
```

It's pretty clear that these instructions

```
WRAX 40,0.0
RDAX 40,1.0
```

can be replaced by:

```
WRAX 40,1.0
```

This saves one instruction. Doesn't save a register yet because the register is also used by the RDFX instruction.

So, how do we find these patterns in general, to modify them? My initial thought is something like this:

Iterate through the generated code from first to last instruction.

Case 1:
If there is a pair of instructions of the form:

```
index n:    wrax reg, 0.0
index n+1: rdax reg, 1.0
```

AND there is no reference to "reg" by any instruction elsewhere in the entire program, then these two instructions can safely be eliminated, and the allocation of **reg** can be eliminated.

Saves two instructions and one register.

Eliminating the allocation of certain registers after the fact will be interesting.

Case 2:
If there is a pair of instructions of the form:

```
index n:    wrax reg, 0.0
index n+1: rdax reg, 1.0
```

This time, there IS a reference to "reg" by another instruction elsewhere in the program.

Replace these instructions with:

```
wrax reg,1.0
```

Saves one instruction.

Case 3:
If there is a pair of instructions of the form:

```
index n:    wrax reg, 0.0
index n+1: rdax reg, inputGain
```

where inputGain is either a variable reference or a constant other than 1.0, these two instructions can be replaced by:

```
sof inputGain, 0.0
```

Saves one instruction.

Top
- Edit post (./posting.php?mode=edit&f=42&p=1675)
- Delete post (./posting.php?mode=delete&f=42&p=1675)
- Report this post (./report.php?f=42&p=1675)
- Warn user (./mcp.php?i=warn&mode=warn_post&f=42&p=1675&sid=d6d1dcc4501596550704606791e62b4c)
- Information (./mcp.php?i=main&mode=post_details&f=42&p=1675&sid=d6d1dcc4501596550704606791e62b4c)
- Reply with quote (./posting.php?mode=quote&f=42&p=1675)

Re: Optimizing Spin ASM code generation (#p1675)

by **disasterarea** » Thu Jun 26, 2014 2:16 pm

The great thing about the FV-1 is that as long as you don't run out of instruction space there are no performance hits for long and loopy code. Of course, your goal here is probably to make sure we don't run out of space (and registers) for no reason, and I think it's great.

The one complaint I have about SPCD is that the generated output is not very readable. That makes it hard to do a quick tweak to the finished code, you have to go back in SPCD and tweak, export again, etc. Even naming POT0, POT1, ADCR, DACL, etc would make things a lot easier to read. Those labels don't require EQU / assembler directives for SpinASM to parse, as far as I know.

Speaking of EQUs, do we have a penalty for using mnemonics? I mean, naming the registers that store the output of each block wouldn't seem to be that difficult. I'm not doing the work here so it's all pie in the sky, but it would be great to have some idea of which register holds which value.

Top
- Edit post (./posting.php?mode=edit&f=42&p=1676)
- Delete post (./posting.php?mode=delete&f=42&p=1676)
- Report this post (./report.php?f=42&p=1676)
- Information (./mcp.php?i=main&mode=post_details&f=42&p=1676&sid=d6d1dcc4501596550704606791e62b4c)
- Reply with quote (./posting.php?mode=quote&f=42&p=1676)

Re: Optimizing Spin ASM code generation (#p1676)

by **Digital Larry** » Thu Jun 26, 2014 3:40 pm

You know, I've been looking at those raw numbers for so long now that I can just tell you off the top of my head that 16 = POT1, ADCL = 20, etc. But that's not something to wish on anyone else.

I'll give this some thought. It's probably not that hard, however consider that there is a variable called "output" in most blocks (I do in fact use names in my source code for creating the blocks). The variable names would wind up being called something like:

SOF5_output (output of block #5 which happens to be a SOF block)
ThreeTap7_output (output of block #7 which is a ThreeTap block)

I could also more liberally comment the code. All I've done up to now is throw out a comment when the code generation for the block starts, so at least you can see where the blocks stop and start.

Top
- Edit post (./posting.php?mode=edit&f=42&p=1961)
- Delete post (./posting.php?mode=delete&f=42&p=1961)
- Report this post (./report.php?f=42&p=1961)
- Information (./mcp.php?i=main&mode=post_details&f=42&p=1961&sid=d6d1dcc4501596550704606791e62b4c)
- Reply with quote (./posting.php?mode=quote&f=42&p=1961)

Re: Optimizing Spin ASM code generation (#p1961)

by **Digital Larry** » Wed Nov 05, 2014 10:46 am

Going back to the fundamental problem of HOW DO WE KNOW whether a register is used farther down the program when we first scan it? And the answer is, we don't. But, we can look for references to all registers first, fill an array, and then scan the instruction list again for code generation. Sounds awesome!

Let's try it.

Now you could RDAX multiple times from a given port, such as any of the pots or the input, and do things with them. The fundamental issue is to determine if a given WRAX and RDAX pair can be eliminated.

So first, scan through the instruction list. There should be only one RDAX for a given WRAX, it should be the next instruction, and they need to be exactly in the form:

```
WRAX regX, 0.0
RDAX regX, 1.0
```

In the instruction line, scan 3 variables from the instruction: inst, reg, scale
You might only get 1 or 2 so you'll have to count how many things came back.
It's best to have this loop only process one line at a time and use the findReg variable to indicate whether we're trying to match an RDAX with a previous WRAX.

```
int count = getLine(inst, reg, scale);
static int findReg;

if(findReg != -1) {
    if(count == 3) {
        if (inst == "rdax" && scale == 1.0)

... work in progress......

if (count == 3) {
    if (inst = "wrax") {
        if (scale == 0.0) {
             findReg = reg;
        }
    }
}

else {
       findReg = -1;
}
```

Top

Display posts from previous: All posts Sort by Post time Ascending Go

Post a reply

4 posts • Page **1** of **1**

Return to Developer's Corner

Jump to: Developer's Corner Go

Quick-mod tools: Lock topic Go

Who is online

Users browsing this forum: Digital Larry and 0 guests

Administration Control Panel