Register Transfer Language: A Primer

Register Transfer Language (RTL) is often used by designers to describe the functionality of a system component in terms of transfers between registers within that system. Modern *hardware description languages* (HDLs) often incorporate RTL as part of the permissible *design flow* of the language proper.

Different people may have different ideas about what RTL is. These differences arise depending upon the backgrounds and engineering roles of the people involved. In this document we will describe one form or RTL that is quite common. Be advised, however, that different forms of RTL exist and you will eventually encounter other forms as you explore this area.

Transfers

Transfers are made in a right-to-left fashion, as assignments are in accomplished in most high-level programming languages. Sometimes people use the terms "transfer" and "assignment" as synonyms when describing RTL. Different variants of RTL use different operators to denote transfers. In our variant we use the <= operator to denote a transfer. As an example, the following RTL fragment

```
TCCR <= 0x3C
TCDR <= R4
```

transfers the literal value `0x3c` to an entity named `TCCR`, and then transfers the value in the entity named `R4` to the entity named `TCDR`.

Transfers in our form of RTL are *blocking*, meaning that each transfer (and all side effects of the transfer) must complete before the next transfer is started. This is akin to how assignments work in most high-level programming languages. This is why, in the fragment listed above, the transfer to `TCCR` happens **before** the transfer to `TCDR`. Some variants of RTL have both blocking and *non-blocking* transfers. Non-blocking transfers allow two or more transfers to occur simultaneously.

Vector Indexing

Vector indexing is used in RTL to indicate the particular segment of an aggregate that is being referenced. You have seen this notation previously when we developed the RTL for the FETCH phase of the atmega328 instruction processing cycle. By way of reminder, the expression that we developed in the FETCH phase was

```
PMDR <= PMEM [ PMAR ]
```

which indicates that the value in program memory (PMEM) at the index indicated by the value of the program memory address register (PMAR) is transferred to the program memory data register (PMDR). In this example the [ ] operator is used to *index into* the program memory, much in the same way that this same operator is used to index into an array in high-level programming languages such as C and C++.

Bit-Slicing

Bit-slicing is a form of vector indexing used in RTL to transfer segments of values from one entity to another. We have covered bit-slicing in lecture. As a reminder, the following RTL fragment

```
DDRB[7:5] <= R16[2:0]
```

transfers three bits from R16 at positions 2 through 0 inclusive to DDRB at positions 7 through 5 inclusive. This could be written as three separate one-bit transfers as follows:

```
DDRB[7] <= R16[2]
DDRB[6] <= R16[1]
DDRB[5] <= R16[0]
```

Concatenation

Concatenation is used in RTL to form one value from one or more segments of other values. Concatenation in our form of RTL is indicated using the { } operator. For example, the following RTL fragment

```
R16 <= { R31[3:0], R30[7:4] }
```

creates a single 8-bit value by concatenating the lower *nibble* (4 bits) of the entity named R31 with the upper nibble of the entity named R30. This 8-bit value is then transferred to the entity named R16. Any number of comma-separated operands can appear inside the { } operator. Please note that the RTL fragment appearing above is equivalent to the following:

```
R16[7:4] <= R31[3:0]
R16[3:0] <= R30[7:4]
```

Comments

In our form of RTL, comments begin with a semicolon (;) and continue to the end of the line. Comments are used to communicate with your colleagues and possibly yourself at some time in the future. Informative comments are always a Good Idea.

An Example

The following example is one way of encoding the logic of the atmega328 ADD instruction in RTL. Each line of RTL is numbered for the sake of clarity and discussion, but these numbers are not part of the RTL itself. For the sake of simplicity in the example below, we do not show what happens in each phase of the instruction processing cycle of the atmega328, but just partition the instruction into a ”FETCH” and ”EXECUTE” phase.

```
FETCH:
    1. PMAR <= PC
    2. assert READ of PMEM
    3. PMDR <= PMEM[PMAR]
    4. IR <= PMDR
    5. PC <= PC + 1
EXECUTE:
    6. IF IR[15:10] = 000011 ; means we're going to perform an ADD
    7. ALU1 <= R[IR[8:4]]    ; copy one value to the ALU
    8. ALU2 <= R[{IR[9],IR[3:0]}] ; copy the other value to the ALU
    9. assert ADD of ALU          ; tell the ALU to do an ADD
    10. R[IR[8:4]] <= ALURESULT   ; transfer the result to the dest reg
```

Lines 1 through 5 above should look very familiar. They are the RTL for the FETCH phase of the atmega328 instruction processing cycle. Line 6 above represents the decoding of the instruction. Lines 7 through 10 above are the RTL that accomplishes the ADD of Rr to Rd via the ALU as specified in the atmega328 Instruction Set Reference.

In line 7 above, the value in the `IR` at bits 8 through 4 is used to index into the 32 general-purpose registers (GPRs) of the atmega328. If, for instance, the IR contained the value 0000110000100011, then line 7 above would take the slice 00010 (bits [8:4]) and use this value (2) as an index into the GPRs. So Line 7 above would transfer the value in R2 into the entity ALU1. The ALU1 and ALU2 entities are internal registers of the ALU that are used to hold operands. The ALURESULT is an internal register that holds the result of the last ALU operation performed.

In Line 8 above, the 1-bit value in IR[9] is concatenated to the 4-bit value in IR[3:0] to create one 5-bit value that is then used to index into the GPRs as we did in line 7. For instance, if the IR contained the value 0000110000100011, then line 8 above would take IR[9] and concatenate to it IR[3:0] to create the 5-bit value 00011 as shown below

```
0 0011
| ----
|  |
|  +-- IR[3:0]
+-- IR[9]
```

This 5-bit value (3) is then used to index into the GPRs. So the RTL on line 8 will transfer the value in R3 into the entity named ALU2.

In this example we do not calculate the condition codes of the status register (SREG) as the actual RTL for the atmega328 ADD instruction would.