Metropolis TTL Platform: User's Manual

Alessandro Pinto
University of California at Berkeley
545P Cory Hall, Berkeley, CA 94720
apinto@eecs.berkeley.edu

September 14, 2004

Contents

Abstract

This document describes the task transaction level (TTL) platform provided by the Metropolis infrastructure. The platform is implemented around a bounded FIFO and its communication protocol. Reading and writing operation are token-based. A medium can be configured with respect to the token size and the FIFO length. Four services are provided to access data stored in a FIFO: *claim_data*, *claim_space*, *release_data*, *release_space*.

1 Communication Model

The TTL platform provides the implementation of a communication channel for modeling systems at task transition level (TTL) [**?**]. The model was originally developed by Philips and it is meant to refine the YAPI model. TTL platform provides four methods to access data:

- `claim_data` is used to check if there is at least a tokens available in a FIFO. This method is blocking;
- `release_space` is called after a token has been read from a FIFO. This method tells the medium that the token has been received and the space can be made available for other tokens;
- `claim_space` is used by a writer to check if there is enough space to write a token in the FIFO. This method is also blocking;
- `release_data` is used to write a token on the output channel after claim_space has return a positive answer.

A typical use of this method is the following: process claims data from the input channel, claims space on the output channel and finally does its computation. When the computation terminates, the result is written on the output buffer using the release_data service and space is released on the input channel.

In order to allow pipelining a number of initial tokens can be provided for each channel.

2 The Bounded FIFO

The TTL platform defines interface methods to access data stored in a FIFO:

```
template(T)
public interface boundedfifooutinterface extends boundedfifointerface-<T>- {
    eval int claim_space();
    eval int query_space();
    eval int guard_query_space();  // added by YW
    update void release_data(T data);
    update void release_data(T[] data);
}
template(T)
public interface boundedfifoininterface extends boundedfifointerface-<T>- {
    eval T claim_data();
    eval void claim_data(T[] data);
    eval int query_data();
    eval int guard_query_data();
    update void release_space();
}
```

Interfaces are templates defining services to store and retrieve data. The meaning of each function has been explained in section 1. The non-blocking version of claim_space and claim_data are query_space and query_data. They return respectively the number of available spaces and tokens in a bounded FIFO.

The TTL bounded FIFO medium is described as follows:

```
template(T)
public medium boundedfifo implements boundedfifoininterface-<T>-,
                                     boundedfifooutinterface-<T>-,
                                    ttlwri,ttlrdi {
  parameter int tokensize;
  parameter int numberoftokens;
  T[] FIFO;
  int rp,wp;
  int ntokens;
  int DataSize;

  public boundedfifo(String n,int ts,int nt,int it, int ds){
    super(n);
  }
  ...
}
```

The $boundedfifo$ medium is a template that implements the input and output interfaces already defined. The constructor takes the following list of parameters: the token size ts, the maximum number of tokens nt, the initial

number of tokens in the FIFO it and finally the data size ds. The internal buffer $FIFO$ is implemented as a circular buffer of which rp and wp are respectively the read and write pointer.

2.1 Interfaces with YAPI processes

The TTL platform has been developed as a refinement of the YAPI platform. YAPI processes communicate through unbounded FIFOs. In order to be able to refine each yapi channel into a bounded FIFO, two adaptation interfaces have to be implemented: one that implements a yapi write using claim_space and release_data and another interface that implements yapi read using claim_data and release_space. These two interfaces are $yapi2boundedfifo$ and $boundedfifo2yapi$.

The two medium implementing these interfaces are respectively $yapi2TTLchannel$ and $TTL2yapichannel$. Since a yapi process could write a big amount of data in a single write, and since the data have to be transferred to the yapi process that is waiting for the, a protocol to exchange data on using the bounded FIFO has to be used.

TTL platform provides a netlist that assemble the two yapi interfaces, the bounded FIFO and the medium that implements the data exchange protocol.

The netlist code looks as follows:

```
package metamodel.plt.TTLtemplate;
import metamodel.plt.yapitemplate.*;
template(T)
public netlist TTLmediumnetlist {
  boundedfifo-<T>- bf;
  rdwrthreshold rwtr;
  yapichannel-<T>- channel;
  yapi2TTLchannel-<T>- y2bf;
  TTL2yapichannel-<T>- bf2y;
  public TTLmediumnetlist(String n,yapichannel-<T>- ch,int ts,int nt,int it,int ds,stfunc f
    super(n);
    channel = ch;
    y2bf = new yapi2TTLchannel-<T>-(n+"y2bf",ts,nt,f);
    bf2y = new TTL2yapichannel-<T>-(n+"bf2y",ts,nt,f);

    rwtr = new rdwrthreshold(n+"rdwrth");

    bf = new boundedfifo-<T>-(n+"bf",ts,nt,it, ds);

    addcomponent(y2bf,this,n+"y2bf_inst");
    addcomponent(bf2y,this,n+"bf2y_inst");
```

```
    addcomponent(rwtr,this,n+"rdwrth_inst");
    addcomponent(bf,this,n+"bf_inst");
    addcomponent(ch,this,n+"yapirefch_inst");

    connect(y2bf,tofifo,bf);
    connect(bf2y,fromfifo,bf);
    connect(y2bf,fromfifo,bf);
    connect(bf2y,tofifo,bf);

    connect(y2bf,rdwrth,rwtr);
    connect(bf2y,rdwrth,rwtr);

    refine(ch,this);
    refineconnect(this,
                  getnthconnectionsrc(ch,yapioutinterface-<T>-,0),
                  getnthconnectionport(ch,yapioutinterface-<T>-,0),
                  y2bf);
    refineconnect(this,
                  getnthconnectionsrc(ch,yapiininterface-<T>-,0),
                  getnthconnectionport(ch,yapiininterface-<T>-,0),
                  bf2y);
  }
}
```

The constructor takes the following set of parameters: ch is the yapichannel to refine, f is s special function to decide the properties of the communication protocol. All the other parameters are used to configure the bounded FIFO. The netlist structure is shown in figure 1

The yapichannel ch is refined using the keyword $refine$ and connection of the original processes to ch are redirected to the two interfaces $y2bf$ and $bf2y$.

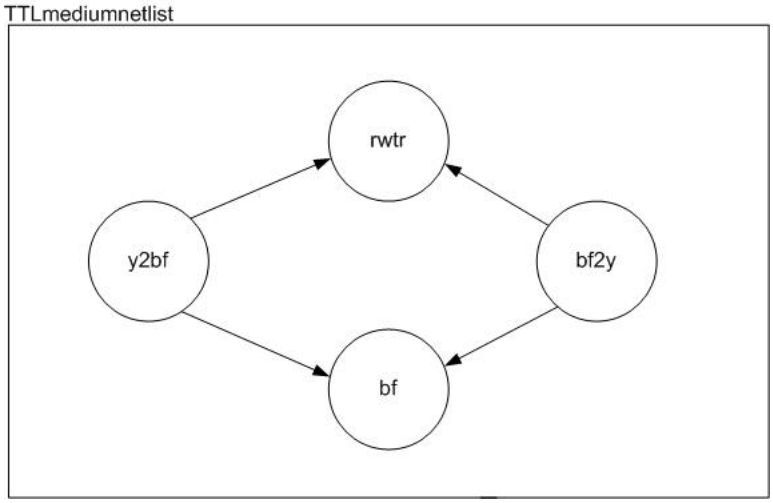

Figure 1: TTL netlist