GWN: a Framework for Packet Radio and Medium Access Control in GNU Radio

Víctor González-Barbone, Pablo Belzarena, Federico Larroca,
Martín Randall, Paola Romero and Mariana Gelós
{vagonbar, belza, flarroca, mrandall, paolar,
mariana.gelos}@fing.edu.uy

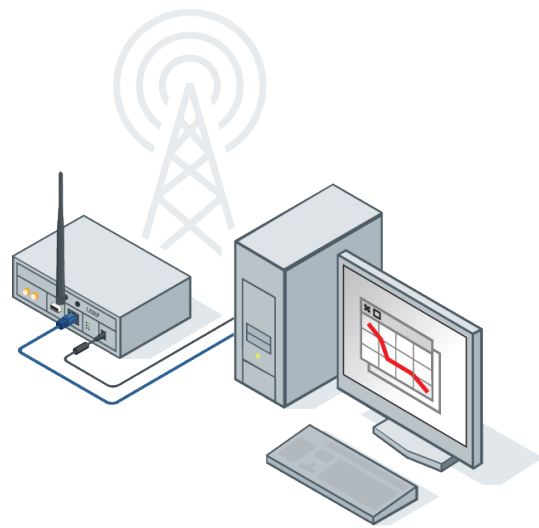
Instituto de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de la República
Uruguay

Wireless Innovation Forum Conference on Wireless Communications
Technologies and Software Defined Radio (WInnComm '17)
San Diego, California

November 16th 2017

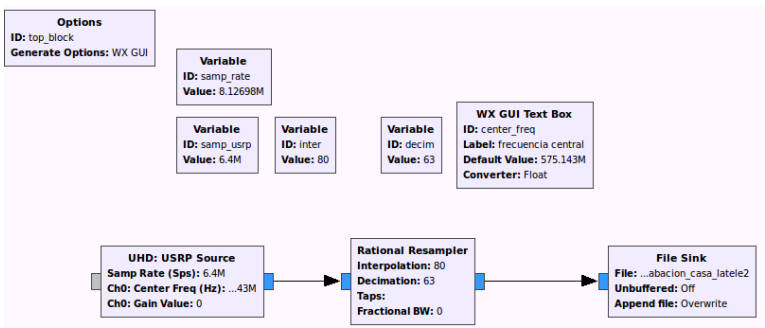
Software Defined Radio (SDR)

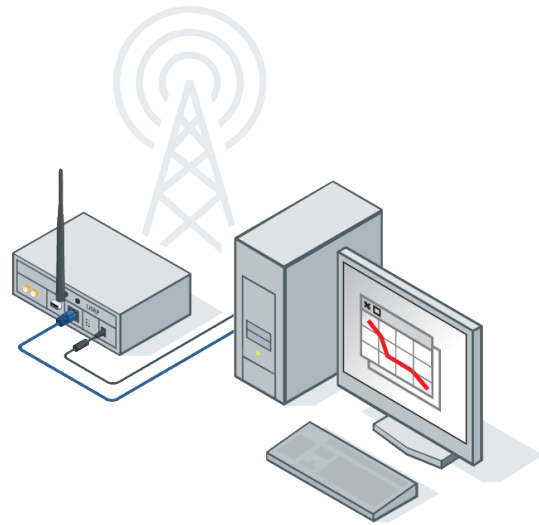
- Basic idea: implement as much as possible of receiver and/or transmitter in software
- We focused on PC-based SDRs: enough sampling rate for most applications + only some hundreds dollars



Software Defined Radio (SDR)

- Basic idea: implement as much as possible of receiver and/or transmitter in software
- We focused on PC-based SDRs: enough sampling rate for most applications + only some hundreds dollars
- Software: GNU Radio
  - Free and open-source SDR development toolkit
    - Provides framework to implement and interconnect blocks
  - Extremely popular
    - Supports most (if not all) SDR hardware
    - Growing base of already-implemented blocks





GNU Radio and Wireless Networks

- GNU Radio was originally conceived for stream-oriented communications

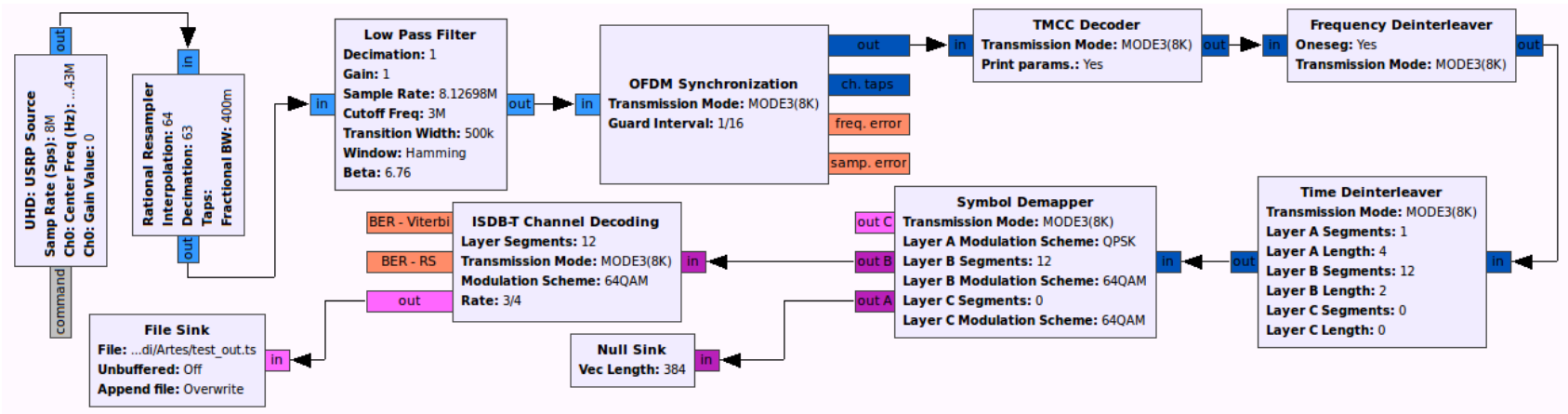


Fig: An example flowgraph of gr-isdbt, our ISDB-T (DTV) receiver

- What if I want to work with packets?

GNU Radio and Wireless Networks

What if I want to work with packets on GNU Radio?

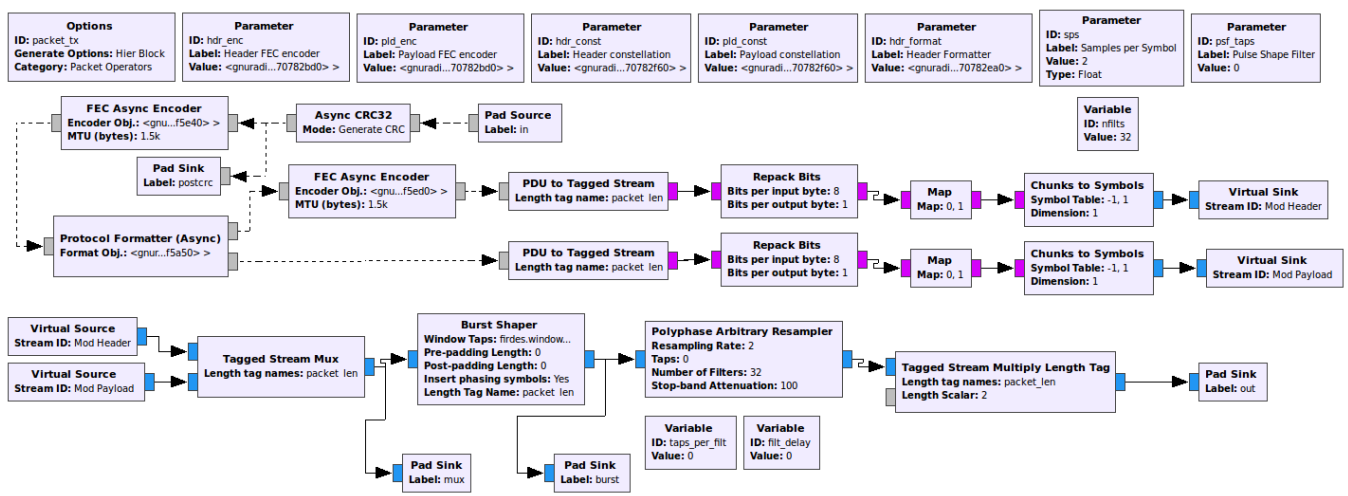


Fig: An example flowgraph of gr-digital (packet_tx.grc)

GNU Radio and Wireless Networks

What if I want to work with packets on GNU Radio?

- **Tagged Streams**
  - Specific samples may be "marked" with extra data (i.e. where a new packet begins)
- **Message passing**
  - Blocks may send and receive **messages**
  - They work asynchronously and separated from the data stream
  - Blocks may even receive messages from external apps
  - Messages are actually PMTs: generic data containers which may contain mostly anything, and in particular packets

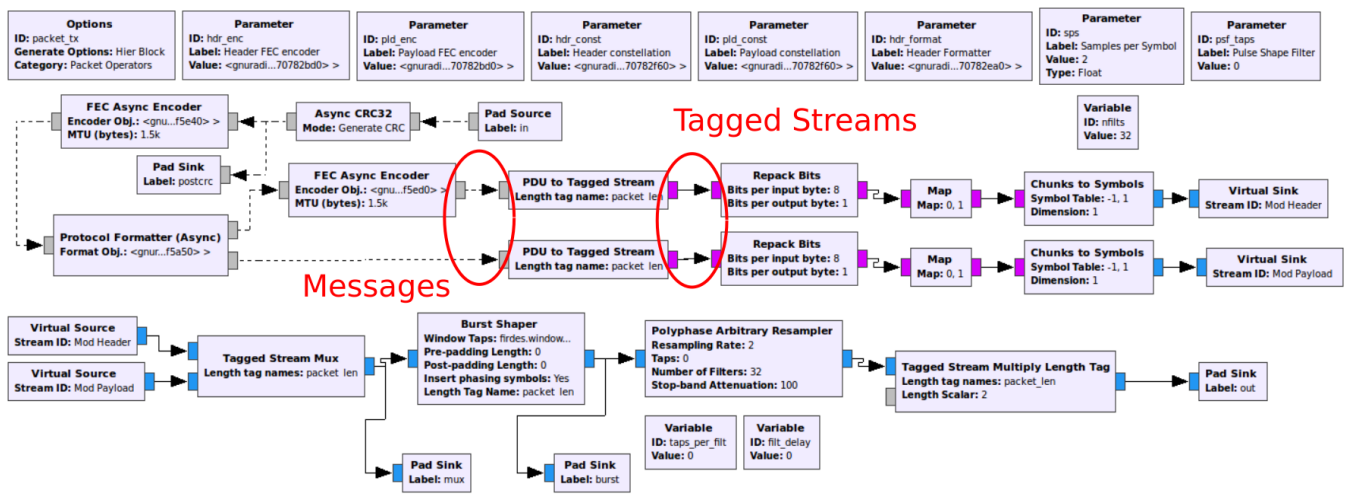


Fig: An example flowgraph of gr-digital (packet_tx.grc)

GNU Radio and Wireless Networks

Natural next step: What if I want to implement the Data Link Layer on GNU Radio?

- What's necessary?
  - TX: Packets have to be sent when certain conditions are met (e.g. free channel, slotted time), acknowledgements waited for a certain time, packets re-sent, etc.
  - RX: As packets are received, error checking/correction is performed, re-ordering may be necessary, (n)acknowledgments sent, etc.
  - The above includes the **Medium Access Control** mechanism

GNU Radio and Wireless Networks

Natural next step: What if I want to implement the Data Link Layer on GNU Radio?

- What's necessary?
  - TX: Packets have to be sent when certain conditions are met (e.g. free channel, slotted time), acknowledgements waited for a certain time, packets re-sent, etc.
  - RX: As packets are received, error checking/correction is performed, re-ordering may be necessary, (n)acknowledgments sent, etc.
  - The above includes the **Medium Access Control** mechanism

What's necessary?

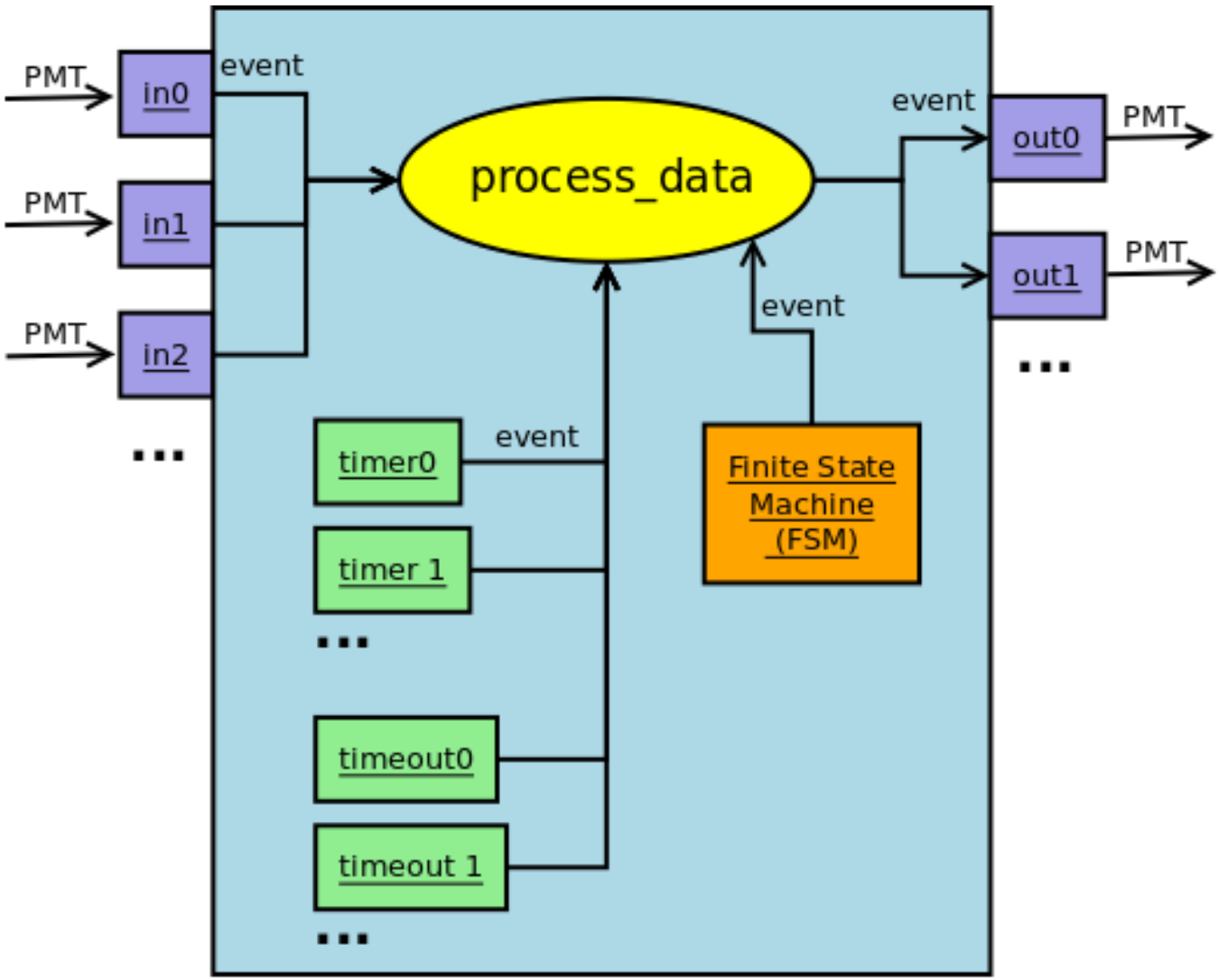
1. A Finite State Machine (FSM) to implement the protocol's logic
2. Events to drive the FSM
3. Timing

GNU Radio Wireless Networks

- We present GNU Radio Wireless Networks (GWN)
- It introduces `gwnblock`: an extension of GNU Radio's basic block which includes
  - An implementation of a FSM, which may be specified very easily
  - Events
  - The possibility to handle time
- It is thus fully compatible with GNU Radio and integrates seamlessly
- We demonstrate it by discussing an ARQ (Automatic Repeat ReQuest) implementation

GWN Architecture

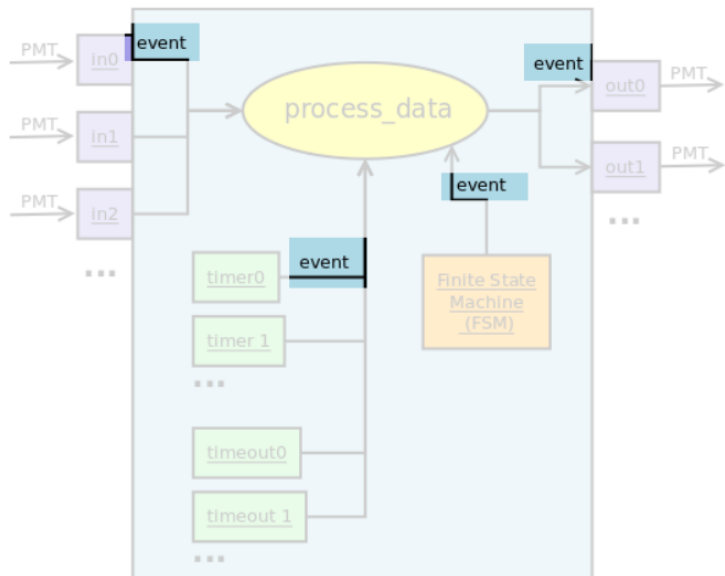
A typical GWN block:



GWN Architecture

Events

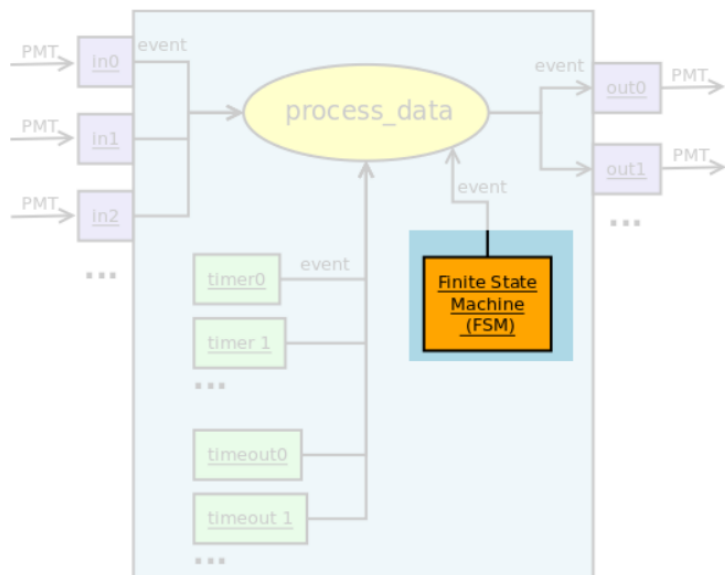
- Events are characterized by:
  - a `Nickname` which indicates the type of Event
  - a `Dictionary` with additional information
- Specialized Events are provided: `EventConfig`, `EventTimer` and `EventComm`, along with constructor functions
  - example: `EventComm` represents an incoming packet, and as such includes source/destination addresses and payload



GWN Architecture

Finite State Machine

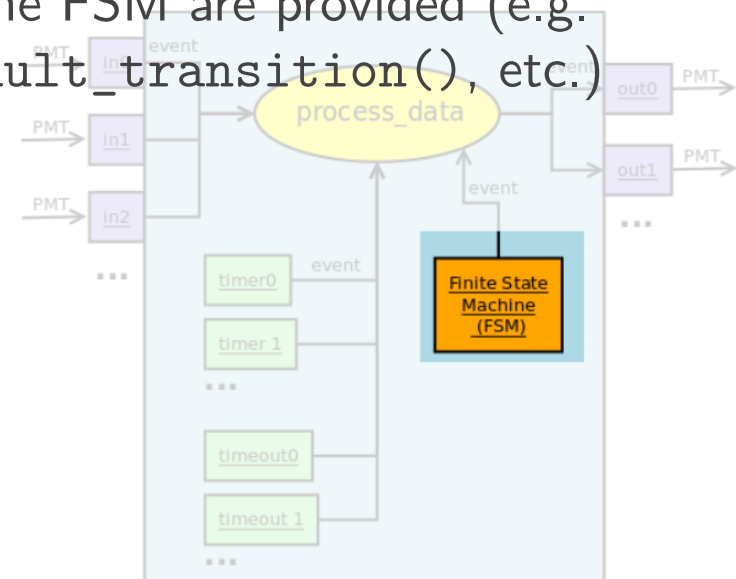
- In addition to states and transitions, GWN's FSM includes:
  - `Action`: a function to be executed on a transition
  - `Memory`: which may be handled in the action functions
  - `Conditions`: a function which, if evaluates to False, the Action and the transition are not executed



GWN Architecture

Finite State Machine

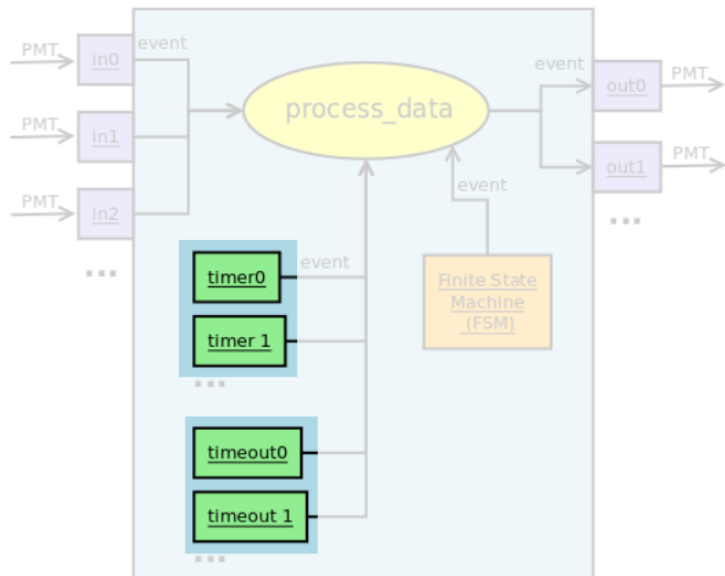
- In addition to states and transitions, GWN's FSM includes:
  - `Action`: a function to be executed on a transition
  - `Memory`: which may be handled in the action functions
  - `Conditions`: a function which, if evaluates to False, the Action and the transition are not executed
- FSM is actually a table of transitions with entries as: `(input,current_state)->(action, next_state, condition)`
- Several methods to modify the FSM are provided (e.g. `add_transition`, `add_default_transition()`, etc.)



GWN Architecture

Handling of Time

- GWN timers generate `Events` (which are processed as any other `Event` by the `process_data()` function)
- An arbitrary number of timers may be attached to a block
- Two timing mechanisms are provided in GWN's current form:
  1. `Timers`: Events are generated periodically for a number of times
  2. `Timeout`: A single Event is generated after a certain time
- All timers may be stopped, reset or interrupted

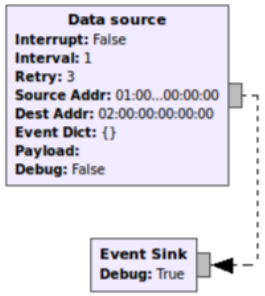


Creating a new block

Let's do a simple example: `Data Source` (it generates data events periodically and sends them out as GNU Radio messages)

1. New blocks are generated (as usual) through `gr_modtool`:

```
gr_modtool add -t sync -l python
```



Creating a new block

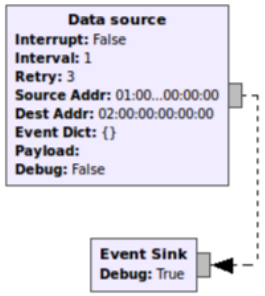
Let's do a simple example: `Data Source` (it generates data events periodically and sends them out as GNU Radio messages)

1. New blocks are generated (as usual) through `gr_modtool`:

```
gr_modtool add -t sync -l python
```

2. Modify the code so that it inherits from `gwnblock`

```
class data_source(gwnblock):
```

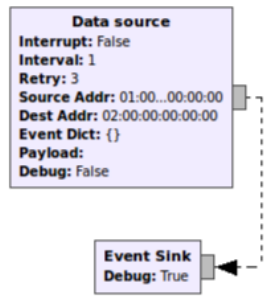


Creating a new block

Let's do a simple example: `Data Source` (it generates data events periodically and sends them out as GNU Radio messages)

1. New blocks are generated (as usual) through `gr_modtool`:

```
gr_modtool add -t sync -l python
```



2. Modify the code so that it inherits from `gwnblock`

```
class data_source(gwnblock):
```

3. Write the constructor (note that the number of timers is specified here):

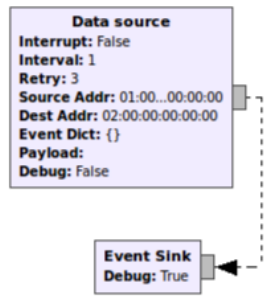
```
def __init__(self, interrupt=False, interval=1.0, retry=5,
     src_addr='', dst_addr='', payload='', ev_dc={}, debug=False
      ):
    # invocation of ancestor constructor
    gwnblock.__init__(self, name='data_source', number_in=0,
         number_out=1, number_timers=1)
```

Creating a new block

Let's do a simple example: `Data Source` (it generates data events periodically and sends them out as GNU Radio messages)

1. New blocks are generated (as usual) through `gr_modtool`:

```
gr_modtool add -t sync -l python
```



2. Modify the code so that it inherits from `gwnblock`

```
class data_source(gwnblock):
```

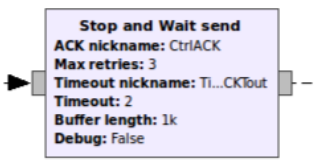
3. Write the constructor (note that the number of timers is specified here):

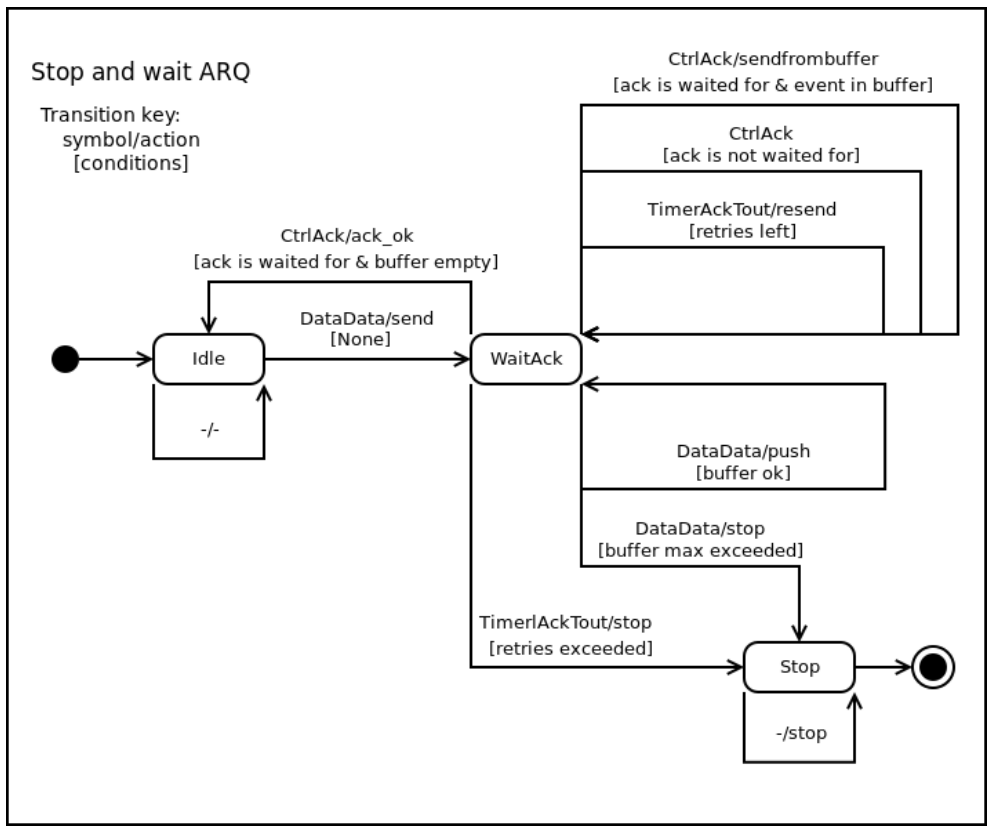
```
def __init__(self, interrupt=False, interval=1.0, retry=5,
    src_addr='', dst_addr='', payload='', ev_dc={}, debug=False
     ):
    # invocation of ancestor constructor
    gwnblock.__init__(self, name='data_source', number_in=0,
        number_out=1, number_timers=1)
```

4. Write the `process_data` function to generate a new `Data Event` and write it on the `Event Port`

Creating a new block

Let's do a more advanced example: an ARQ Stop and Wait transmitter (it doesn't send a new packet until an acknowledgement for the previous one was received, and a timeout generates a re-transmission)

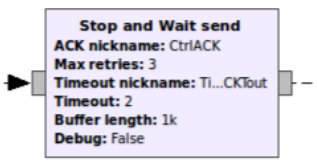




Creating a new block

Let's do a more advanced example: an ARQ Stop and Wait transmitter (it doesn't send a new packet until an acknowledgement for the previous one was received, and a timeout generates a re-transmission)

1. Create the new block as before, but include a `timeout` this time:

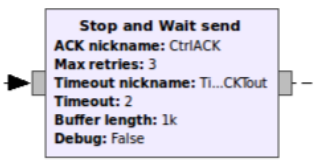



```
def __init__(self, ack_nickname='CtrlACK', max_retries=3,
             tout_nickname='TimerACKTout', timeout=1.0, buffer_len=1000,
              debug=False):
    # invocation of ancestor constructor
    gwnblock.__init__(self, name='stop_wait_send', number_in=1,
                      number_out=1, number_timeouts=1)
```

Creating a new block

Let's do a more advanced example: an ARQ Stop and Wait transmitter (it doesn't send a new packet until an acknowledgement for the previous one was received, and a timeout generates a re-transmission)

1. Create the new block as before, but include a `timeout` this time:




```
def __init__(self, ack_nickname='CtrlACK', max_retries=3,
             tout_nickname='TimerACKTout', timeout=1.0, buffer_len=1000,
             debug=False):
    # invocation of ancestor constructor
    gwnblock.__init__(self, name='stop_wait_send', number_in=1,
                      number_out=1, number_timeouts=1)
```

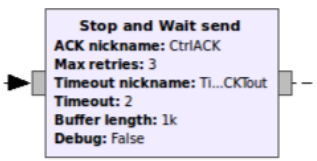
2. We'll use an FSM this time, so `process_data` may include simply two lines:

```
def process_data(self, ev):
    self.fsm.process(ev.nickname, event=ev, block=self)
    return
```

Creating a new block

Let's do a more advanced example: an ARQ Stop and Wait transmitter (it doesn't send a new packet until an acknowledgement for the previous one was received, and a timeout generates a re-transmission)

1. Create the new block as before, but include a `timeout` this time:




```
def __init__(self, ack_nickname='CtrlACK', max_retries=3,
    tout_nickname='TimerACKTout', timeout=1.0, buffer_len=1000,
     debug=False):
    # invocation of ancestor constructor
    gwnblock.__init__(self, name='stop_wait_send', number_in=1,
        number_out=1, number_timeouts=1)
```

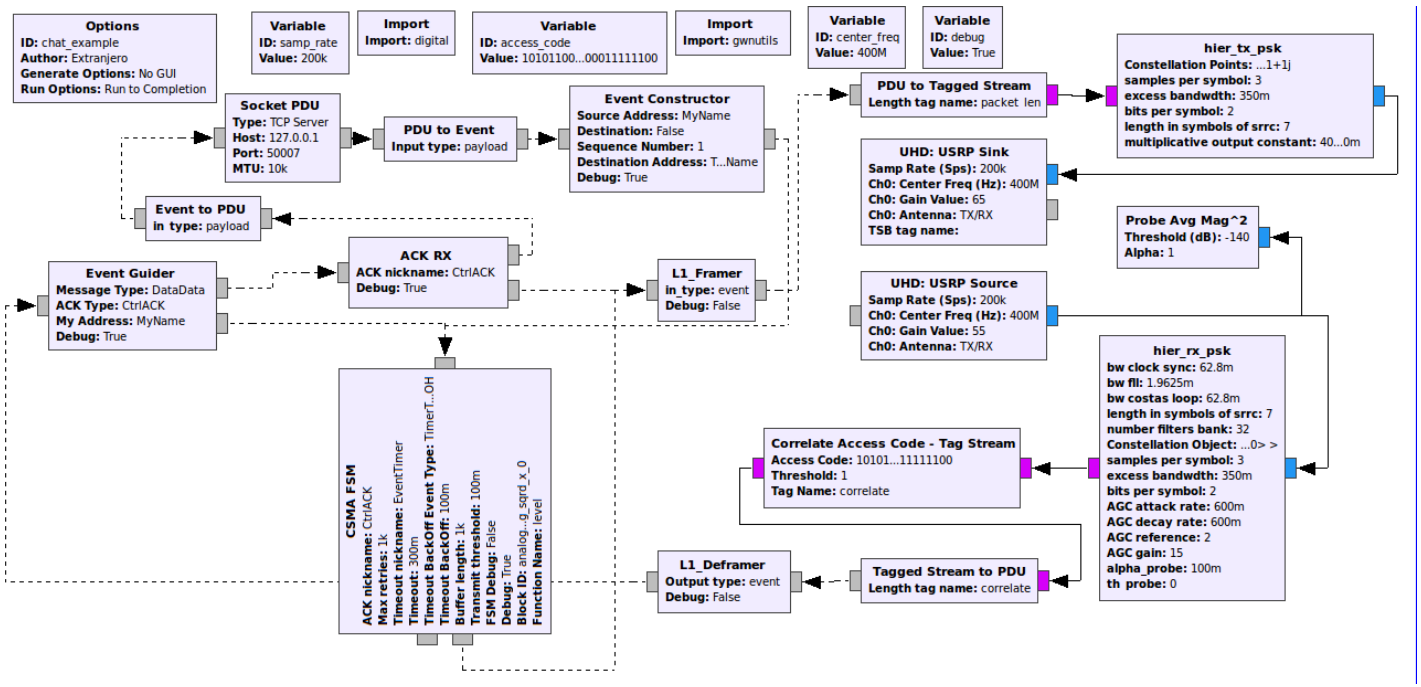
2. We'll use an FSM this time, so `process_data` may include simply two lines:

```
def process_data(self, ev):
    self.fsm.process(ev.nickname, event=ev, block=self)
    return
```

3. Write the FSM. For instance:

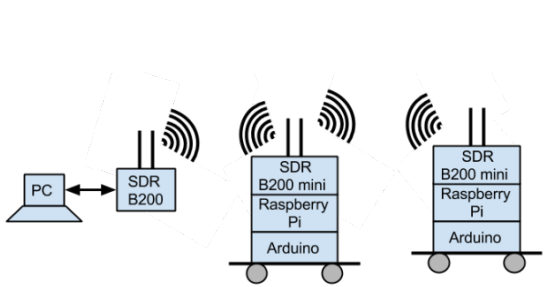
```
def stop_wait_send_fsm(blk):
    # Code to initialize the FSM goes here
    f.add_transition ('TimerACKTout', 'WaitAck', stop, 'Stop', [
        'self.nr_retries > block.max_retries']) # retries
        exceeded
```

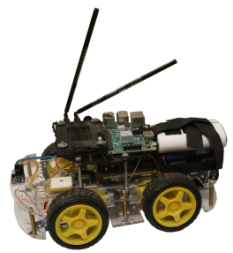
Demo

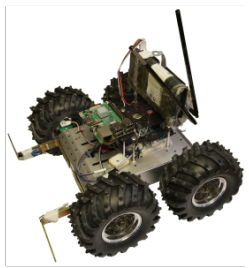

Options
ID: chat_example
Author: Extranjero
Generate Options: No GUI
Run Options: Run to Completion
Variable
ID: samp_rate
Value: 200k
Import
Import: digital
Variable
ID: access_code
Value: 10101100...00011111100
Import
Import: gwnutils
Variable
ID: center_freq
Value: 400M
Variable
ID: debug
Value: True
Socket PDU
Type: TCP Server
Host: 127.0.0.1
Port: 50007
MTU: 10k
PDU to Event
Input type: payload
Event Constructor
Source Address: MyName
Destination: False
Sequence Number: 1
Destination Address: T...Name
Debug: True
PDU to Tagged Stream
Length tag name: packet_len
hier_tx_psk
Constellation Points: ...1+1j
samples per symbol: 3
excess bandwidth: 350m
bits per symbol: 2
length in symbols of srrc: 7
multiplicative output constant: 40...0m
UHD: USRP Sink
Samp Rate (Sps): 200k
Ch0: Center Freq (Hz): 400M
Ch0: Gain Value: 65
Ch0: Antenna: TX/RX
TSB tag name:
Event to PDU
in type: payload
Probe Avg Mag^2
Threshold (dB): -140
Alpha: 1
Event Guider
Message Type: DataData
ACK Type: CtrlACK
My Address: MyName
Debug: True
ACK RX
ACK nickname: CtrlACK
Debug: True
L1_Framer
in_type: event
Debug: False
UHD: USRP Source
Samp Rate (Sps): 200k
Ch0: Center Freq (Hz): 400M
Ch0: Gain Value: 55
Ch0: Antenna: TX/RX
hier_rx_psk
bw clock sync: 62.8m
bw fll: 1.9625m
bw costas loop: 62.8m
length in symbols of srrc: 7
number filters bank: 32
Constellation Object: ...0> >
samples per symbol: 3
excess bandwdth: 350m
bits per symbol: 2
AGC attack rate: 600m
AGC decay rate: 600m
AGC reference: 2
AGC gain: 15
alpha_probe: 100m
th_probe: 0
CSMA_FSM
ACK nickname: CtrlACK
Max retries: 1k
Timeout nickname: EventTimer
Timeout: 300m
Timeout BackOff Event Type: TimerT...OH
Timeout BackOff: 100m
Buffer length: 1k
Transmit threshold: 100m
FSM Debug: False
Debug: True
Block ID: analog...g_sqrd_x_0
Function Name: level
Correlate Access Code - Tag Stream
Access Code: 10101...11111100
Threshold: 1
Tag Name: correlate
L1_Deframer
Output type: event
Debug: False
Tagged Stream to PDU
Length tag name: correlate

Conclusions and Future Work

- We've been using GNU Radio for several years now (both education and research), and wished to extend it to support packet communications
- `Event` objects for inter block communication, conversion to and from GNU Radio PMTs, and the integration of `Event` inputs and outputs in a generic GWN block is the result of some years worth of iterations of GWN
- GWN has been used successfully "on the field"

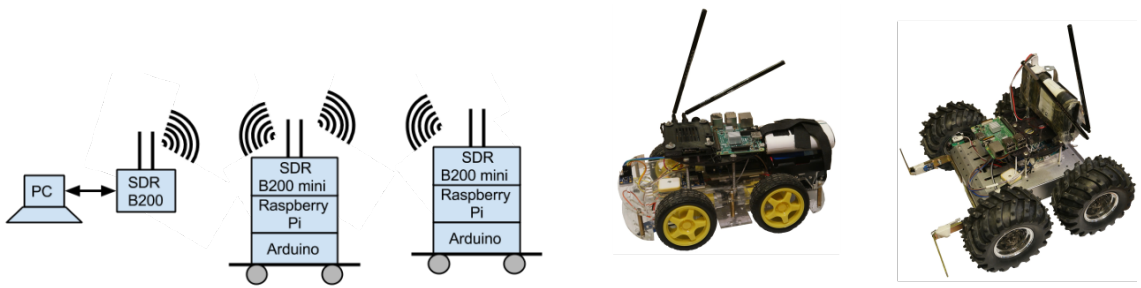






Conclusions and Future Work

- We've been using GNU Radio for several years now (both education and research), and wished to extend it to support packet communications
- `Event` objects for inter block communication, conversion to and from GNU Radio PMTs, and the integration of `Event` inputs and outputs in a generic GWN block is the result of some years worth of iterations of GWN
- GWN has been used successfully "on the field"



- Although the number of protocols already implemented in GWN is still modest, being open and free we welcome contributions!
- Performance is an issue. We are evaluating a transition to a C++ implementation which should be transparent to users

Thanks for your time!

Any questions?

Federico 'Larroca' La Rocca - flarroca@fing.edu.uy

https://github.com/vagonbar/gr-gwn